AFRL-IF-RS-TR-1999-64
Final Technical Report
April 1999

MODEL-BASED HUMAN-COMPUTER INTERACTION

Northwestern University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. B321

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1999-64 has been reviewed and is approved for publication.

APPROVED: Michael J. McHale

MICHAEL MCHALE
Project Engineer

FOR THE DIRECTOR:

NORTHRUP FOWLER, III, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

MODEL-BASED HUMAN-COMPUTER INTERACTION

Lawrence Birnbaum
Ray Bareiss

Contractor: Northwestern University
Contract Number: F30602-94-0219
Effective Date of Contract: 30 June 1994
Contract Expiration Date: 31 December 1997
Short Title of Work: Model-Based Human-Computer Interaction

Period of Work Covered: Jun 94 - Dec 97

Principal Investigator: Lawrence Birnbaum
Phone: (847) 491-3500
AFRL Project Engineer: Michael McHale
Phone: (315) 330-1458

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Lawrence Birnbaum, AFRL/IFTB, 525 Brooks Road, Rome, NY 13441-4505.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1999	Final Jun 94 - Dec 97

4. TITLE AND SUBTITLE

MODEL-BASED HUMAN-COMPUTER INTERACTION

5. FUNDING NUMBERS

C - F30602-94-C-0219
PE - 61101E
PR - B321
TA - 00
WU - 01

6. AUTHOR(S)

Lawrence Birnbaum and Ray Bareiss

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Northwestern University
The Institute for the Learning Sciences
1890 Maple Avenue
Evanston IL 60208

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

Air Force Research Laboratory/IFTB
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-1999-64

11. SUPPLEMENTARY NOTES

Air Force Research Laboratory Project Engineer: Michael McHale/IFTB/(315) 330-1458

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The purpose of this project was to investigate the application of explicit task modeling techniques to human-computer interaction. In the first phase of this project, reported in Part I of this document, the focus of the investigation was on techniques for using task models to direct interface design and automatically compile user interfaces. This resulted in the development of a prototype model-based interface design tool, MODEST. The second half of the project concentrated on techniques for directly employing task models in the architecture of intelligent performance support systems. This resulted in the design and implementation of a fully functioning prototype integrated performance support system in the area of Air Campaign Planning.

14. SUBJECT TERMS

Computers, Software, HCI

15. NUMBER OF PAGES

130

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

Preface

This document constitutes the Final Report for the project "Model-Based Human-Computer Interaction," carried out by Northwestern University's Institute for the Learning Sciences, and supported by the Defense Advanced Research Projects Agency, monitored by Rome Laboratory under contract F30602-94-C-0219.

The purpose of this project was to investigate the application of explicit task modeling techniques to human-computer interaction. In the first phase of this project, reported in Part I of this document (which essentially reproduces the mid-way progress report previously delivered to Rome Laboratory), the focus of the investigation was on techniques for using task models to direct interface design and automatically compile user interfaces. This resulted in the development of a prototype model-based interface design tool, MODEST.

After consultation with the DARPA Program Manager in charge of DARPA's HCI Initiative, Dr. Allen Sears, the project was re-directed after the mid-point to concentrate on techniques for directly employing task models in the architecture of intelligent performance support systems. This resulted in the design and implementation of a fully-functioning prototype integrated performance support system in the area of Air Campaign Planning. The system links to and supports the Air Force's JFACC Planning Tool (JPT). It has been deployed at Rome Laboratory, NY, and at the Air-Ground Operations School, Hurlburt AFB, Florida.

Part I: Model-Based Interface Design

Part I-a: Interface design based on standardized task models

1. Introduction

Current computer software is often complex or arcane to the point where users have a great deal of difficulty understanding it. Indeed, it has been argued that they shouldn't need to understand it: It should understand them instead. The approach that follows from this premise is often termed user modeling, because it entails formulating and maintaining a model of the individual user's goals, knowledge, and dispositions throughout his or her interaction with the system. Unfortunately, while appealing in principle, in practice attempts to model the idiosyncrasies of individual users often result in systems that are difficult to construct and brittle in operation.

We believe that there is a far more cogent and practical approach to achieving fluent man-machine interaction. In fact, it is necessary neither for users to fully understand systems, nor for the systems to understand them: It would be closer to the mark to say system and user must both understand the task at hand. In other words, what needs to be modeled in an interface is not the user, but the task. Acting on the basis of roughly commensurate models of their shared task, human and system can operate "in sync," so that, for example, the system is able to provide timely and relevant options and information, address likely questions and confusions, and quickly and accurately ascertain and perform required actions. By understanding the user's task, the system can make the right decisions in carrying out these and other essential functions of an interface. Shared task models thus provide the context that is necessary to support fluent interaction between human and computer.

To the extent that any software system is capable of carrying out essential interface functions at appropriate times, its design must reflect some model of the user's task. In current interface design practice, however, this model is generally not explicit in the operation of the interface itself, and indeed may not even be explicit in the design process. Typically, a good interface design is the result of a skilled human designer relying on his or her intuitive knowledge of the task in order to make the interface do the right thing at the right time. In the best case, the design process may be iterative, with user testing employed to refine and improve the initial design empirically. Either way, the current state of the art in interface design is such that every interface is custom designed and built, and its quality is totally dependent on the skill of the interface designers.

As with any other engineering design process, improvement in the design methodology for interfaces depends, in large part, upon making explicit what has previously been implicit. Insofar as interface designs depend upon implicit models of the user's task, it is impossible to standardize the creation of such designs. This in turn makes it difficult to communicate and teach the design principles involved, or to automate aspects of the design process.

Our approach aims to overcome these limitations through the use of *explicit, standardized* task models. We are developing a library of such models formulated in terms of a well-defined modeling language. The models that we are building attempt to capture common knowledge about basic cognitive tasks such as *testing, threat detection, planning, communication, process monitoring*, and so on. Models of particular user tasks are developed by combining and parameterizing entries selected from this library of standardized task models. In addition, components of these models are tied to interface constructs that support the user's performance of the task. By tying model components to appropriate interface objects in this way, and combining and parameterizing those objects as the designer combines and parameterizes the corresponding models, it is possible to automatically compile a significant portion of the user interface. Similar approaches have been proposed previously (Puerta *et al.*, 1994; Szekely, Luo and Neches, 1993; Johnson, Johnson & Wilson, 1995).

Using a design tool based on this approach, interface designers conceptualize and design interfaces in terms of the tasks that their systems are meant to support, rather than in terms of interface implementation constructs such as windows, menus, or dialog boxes. Thus, for example, if the user were to specify that the task to be supported by the system under development involved the monitoring of a set of ongoing processes, the system would select an interface framework appropriate for such process monitoring tasks. The tool would then follow up by asking the designer to specify the parameters to be monitored by the user of the system, the safe operating ranges for these parameters, and so on. For each parameter being monitored, the designer would then be asked to select an indicator to be used in displaying the value of that parameter, and an appropriate alarm for signaling the user should the parameter approach the boundary of its safe operating range. Because such concepts as *parameter, safe operating range, indicator*, and *alarm* are explicitly represented in the task model for process monitoring, the designer can conceive of the interface in terms of these functionally meaningful constructs, while leaving the details of implementation to the interface design tool.

An interface design tool incorporating standardized task models can thus capture and exploit commonalities among systems designed to support broad classes of tasks. Using the knowledge associated with these models as background, the system can engage in a focused interaction with the interface designer, allowing him or her to specify features of the interface

in functional terms that relate to the task, rather than in terms of low-level, platform-dependent primitive operations. We have developed a prototype tool, MODEST (Model-based Design Employing Standardized Tasks), based on this.

2. The Model-Based Design Process

Our model-based approach to generating an interface involves three stages: (1) Modeling the current application task by selecting and parameterizing task models from a standardized library. This involves instantiating appropriate sub-tasks and associated conceptual entities for the current application. (2) Identifying specific interface actions (gestures) and multimedia resources to represent those sub-tasks and conceptual entities. (3) Graphically arranging and sizing the interface objects on the screen. MODEST supports all three stages of this process, although it leaves decisions about graphical arrangement to the designer. The designer interacts with the tool through a question-and-answer dialog in the first two stages, and via direct-manipulation in the last.

As an initial design experiment, we have used MODEST to rationally reconstruct the interface for Sickle Cell Counselor, an educational program that involves a simulation in which the student takes a blood sample from a patient, runs it through a gel-electrophoresis machine, and interprets the results to determine the patient's hemoglobin (Bell, Bareiss and Beckwith, 1994).

We have developed and standardized an abstraction of this task called **Sample-Test-Interpret**, which comprises taking a sample of something, testing it for some property, and interpreting the results. Driving down a level, the **Sample** sub-task involves extracting samples (e.g., blood) from one or more sample sites (e.g., a person), perhaps by using some sample extraction device (e.g., a syringe), and placing the samples in a storage container (e.g., a test tube). Similarly, the **Test** sub-task may involve loading a test instrument (e.g., a gel-electrophoresis device) by transporting the sample to the instrument from the storage container using a transport implement (e.g., a dropper) and then turning the instrument on. Finally, the **Interpret** sub-task may involve comparing the output of the test instrument with a visual key of some type, and choosing the best match. The terms listed in italics above correspond to the conceptual entities of the task.

In the first phase of the dialog, the designer chooses appropriate tasks and sub-tasks, and then associates the entities to be manipulated in a particular application (e.g., a syringe) with their corresponding roles in the abstract task model he has chosen (e.g., the sample extraction device). The key point is that the behavior of these entities, in functional terms, is largely determined either by the nature of the entities themselves, or by the nature of the

task at this rather abstract level: A sample extraction device can either be empty or full. It can be filled with a sample, if empty, only when positioned over an appropriate sample site. It can be emptied, if full, only over an empty container. Because these behaviors are determined by the task context and its associated conceptual entities at a high level, it is possible to associate objects and methods to generate these behaviors with the abstract task model. Generating the interface code then becomes a matter of appropriately parameterizing these objects and methods as the designer parameterizes the corresponding model.

In the second phase of the dialog, the designer must associate the entities from the model with media elements that represent them in their different states. For example, what does a syringe look like when it is empty? What does it look like when it is full? Next, he must choose interface idioms to implement actions in the model, e.g., dragging and dropping to represent the movement of an object, or clicking a button to represent starting a process. When all the necessary media resources and interface idioms have been identified, the designer can previews, edit, and run the graphical interface.

3. An Extended Example

To illustrate this model-based design process, we present an extended excerpt of a design session in which MODEST is used to re-create a portion of the interface for Sickle Cell Counselor. The design session begins by choosing an initial abstract task model from a library of tasks:

```
Choose a task to implement
> SampleTestInterpret
```

The main purpose of the Sample-Test-Interpret task is to classify an attribute of a sample of something. In this scenario, the attribute is the hemoglobin genotype of a blood sample, which can take on the values AA, AS, or SS. In many cases, the user cannot specify a task parameter (in this instance, the attribute) simply by selecting an existing object. Instead, the user specifies a name, and MODEST creates a new object to fill the role:

```
What attribute are you trying to determine?
> HemoglobinGenotype
```

The system doesn't know anything about genotypes, blood, or people, but creates an attribute object called "HemoglobinGenotype." It then asks the designer to specify its type:

```
What kind of attribute is HemoglobinGenotype?
> DiscreteAttribute
```

The genotype object is specialized to a "DiscreteAttribute" object.

```
How many values of HemoglobinGenotype are there?
> 3
```

The system proceeds to ask for the names of the three attribute values and creates objects corresponding to each.

```
What will the user be sampling?
> Blood
```

MODEST instantiates the sample variable of the task and gives it the name "Blood." Blood doesn't mean anything to the system yet; it's simply a placeholder.

At this point, the task is decomposed into its three main sub-tasks; the designer can choose to instantiate and refine these sub-tasks in any order. In this example, we will refine the model in the temporal order of the user's task:

```
Pick a subtask to refine.
> Sample
```

In order to refine a task model, it is necessary to know in general terms how the user and system will interact in the performance of the task. One possibility is to construct a direct-manipulation interface to enable the user to carry out the task. Other available alternatives include providing buttons that allow the user to step through the task (like paging through a book), showing the user a movie of the task being performed, or simply omitting the task altogether.

```
How do you want the user to sample the blood?
> DirectManipulation
```

Given that the user will perform the task directly, the system must know more about properties of blood that are relevant to the task. In this case, the distinction that matters in the interface is that blood is a liquid rather than a solid object: It can be poured into containers, but not dragged around.

```
What kind of composition does blood have?
> FluidLike
```

In addition to the sample itself, the sample-extraction task involves three variables: a sample source from which the sample will be extracted, a sample extraction device, and a destination container to hold the extracted sample. Here, the designer creates a new source by typing in the name of a patient, Laron. In the context of a larger program, the entity "Laron" might already exist and the designer would be able to choose it from a menu:

Where does the blood come from?

> Laron

The designer must now indicate an extraction device. MODEST knows about two devices that can be used to extract liquids: syringes and droppers. The designer may choose one of these or create a new device at this time:

What device will be used to extract the blood from Laron?

> Syringe

As with the extraction device, MODEST knows about a couple of types of objects that can be used to contain liquids: test tubes and beakers. (In fact, it also suggests a third possibility, Laron himself.)

Where will the user put the blood after it has been sampled?

> TestTube

At this point, the designer has finished identifying the conceptual entities in the Sample sub-task and proceeds to refine the Test and Interpret sub-tasks in much the same way. Once these have been instantiated and refined, the designer has completed the conceptual phase of the design. MODEST then proceeds to guide the designer in refining the interface itself. Part of this entails requesting graphical depictions of the visible entities in the interface in each of their different states:

What does Laron look like before taking the sample?

> *The designer selects a file containing a picture of Laron.*

In this particular case, Laron does not change appearance when the blood is drawn, so the user specifies the same image in answer to both questions:

What does Laron look like after taking the sample?

> *The designer selects the same picture again.*

The more critical aspect of refining the interface entails determining the primitive interface actions or gestures with which the user will achieve the sub-tasks. These actions are represented as objects called scene managers that encapsulate the control structure for the interface. The library of scene managers includes such user-interface clichés as transporting an object by dragging it, starting a process by clicking on a control and letting it finish automatically, starting and stopping a process, and several others. In this particular case, since blood is a fluid, the appropriate cliché to realize the sample extraction task is the interface action transport an object using a container (in this case, the syringe):

What screen action do you want the user to perform to sample the blood?

> Transport a substance by clicking over a source and clicking over a destination.

The transport by clicking action causes the syringe to be displayed as the mouse pointer. The syringe changes its appearance when it changes state from "empty" to "full" and vice versa. The user must now specify its appearance in these two states:

What does the syringe look like before taking the sample?
> *The designer selects a picture representing an empty syringe cursor.*

Rather than ask what blood looks like, the system simply asks what the sample device looks like when it is full of blood. This is much simpler for fluid samples, but limits the ability to depict different quantities of a sample. It also represents something of a limitation on the generativity of our system, insofar as the same container with differently-colored fluids in it would have to be represented in terms of entirely distinct states of the container.

What does the syringe look like after taking the sample?
> *The designer selects a resource representing a syringe full of blood.*

The dialog proceeds to request empty and full pictures of the test tube, the electrophoresis machine, the test result and genotype key. The designer determines that the machine will be loaded using the same sort of transport action as in the sample task, that the machine will be turned on using a click to start process action, and that the visual comparison will be a simple display. When all the resources and actions are specified, the "Preview" button is enabled, and the designer can construct, lay out, and interact with the resulting interface.

The interface previewer constructs and displays each scene as a window with all of the graphical objects arranged horizontally across the top of the screen. The designer then lays out the interface by sizing and dragging the objects to their desired locations. When he enters "user mode," he can then run the interface to verify its behavior (see Figure 1).

In this case, the interface for the sample task consists of a picture of Laron, an empty test tube, the mouse pointer represented as an empty syringe, and an "exit scene" button. When the user clicks on the picture of Laron with the syringe, the pointer changes to an image of a syringe filled with blood. When the user now clicks on the test tube, the tube fills with blood, and the syringe image is replaced by the standard arrow mouse pointer. The "exit scene" button then becomes enabled, permitting the user to leave the sample extraction scene. If the user were to perform the wrong action, such as clicking on the empty test tube first, nothing will happen, or an error message will be generated, depending upon the designer's preference.

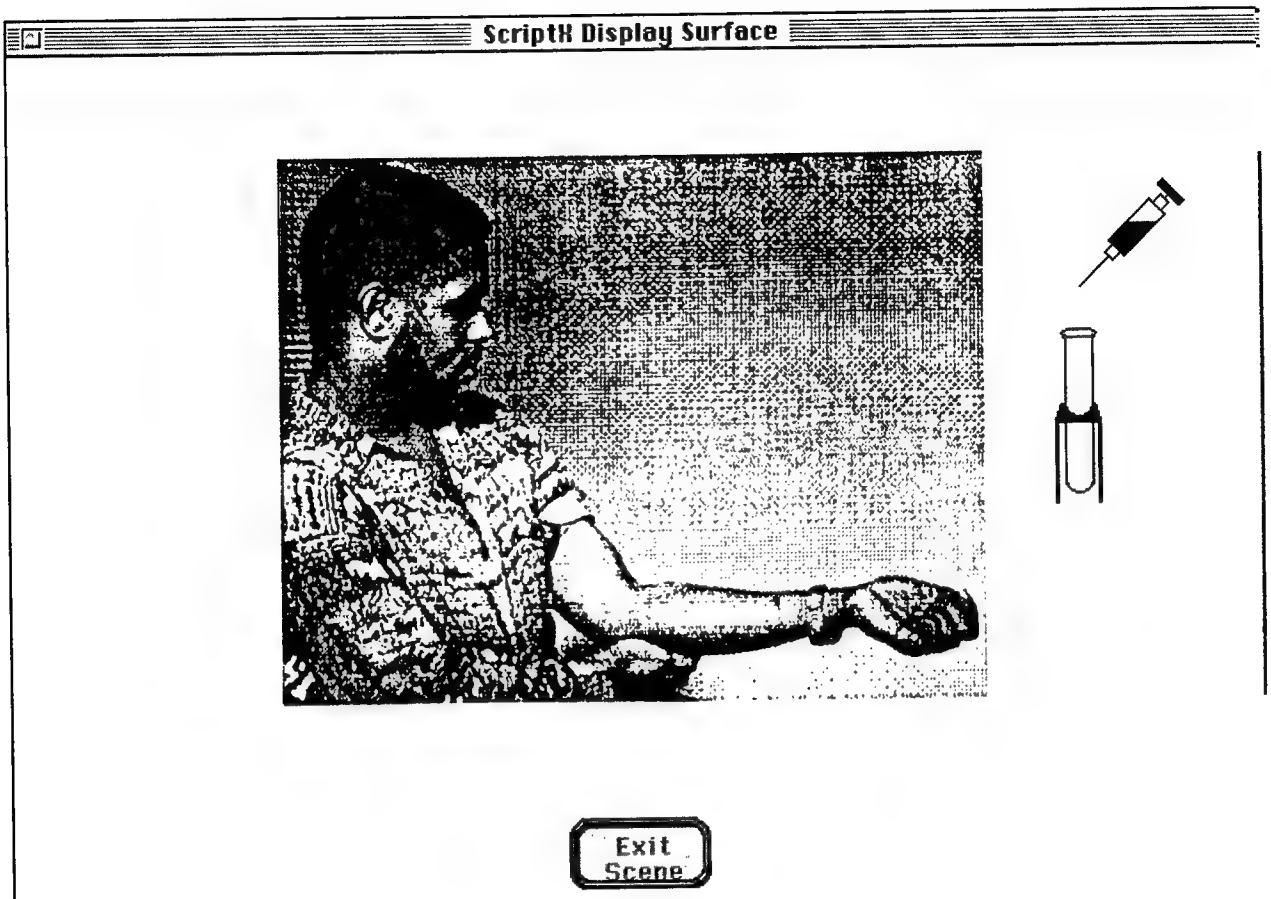

Figure 1. Extracting a blood sample

4. In-context Help & Advice

In addition to guiding the design of an interface, the explicit task model that is embedded in the interface serves an important function in its operation. The model allows the interface to provide contextually relevant help and advice in the vocabulary of the user's task. Rather than simply describing the structural elements of an interface and saying what all the buttons do, a task-driven interface can explain what to do next, why and how to do it. MODEST constructs a palette of three buttons which can be incorporated into the interface: "What next?", "Why?" and "How?" (ref Button Theory). It constructs simple verbal answers to these questions by tracking the user with respect to the task model.

4.1 What Next

One of the major obstacles to interface usability is to knowing what actions are possible. This is especially true of command-line interfaces, but is also a problem with graphical-user interfaces. Sometimes affordances are not obvious, or the user just gets lost. The What next? button determines what tasks can be performed next and describes them to the user in

the context of the current interface. For example, the Sickle Cell Counselor might advise: "Take blood from Laron and store it in the test tube."

4.2 Why

The "Why" button explains the purpose of a task in terms of its role in achieving a higher-level task. For example, the Sickle Cell Counselor might explain: "Take blood from Laron in order to achieve the Sample task."

4.3 How

The "How" button explains the intended method for achieving a task in terms of the subtasks or the individual actions or gestures. For example, the Sickle Cell Counselor might instruct: "Take a blood sample by clicking the syringe over Laron and then clicking over the test tube."

4.4 Error Catching help

Another type of help that was implemented in an earlier version of the system was more proactive in that it responded whenever the user performed some action that was not recognized by the system. For example, if the user clicked on an inactive object, the system would respond with "what next" and "how" information. This tended to annoy users, consequently it was not re-implemented in the second version of MODEST.

5. Representations

There are five main kinds of objects in a task-driven interface:

(1) A *task model* is a functional description of what the user will do and what these actions are trying to achieve.

(2) A *scene manager* is a procedural description of how the user will achieve a task step in the interface in terms of mouse gestures, etc.

(3) A *conceptual entity* is a functional description of an object in the world, such as a container.

(4) A *presenter* is an object that graphically depicts an entity in its different states.

(5) A *widget* (such as a listbox) combines conceptual and presentation aspects into a single object.

The interface itself is constructed as a hierarchy of scene managers which bind conceptual entities as parameters and point back to the task models they implement.

The task modeling language represents the control flow and data flow of the user's task and the interface constructs that implement it. More specifically, the control flow encompasses both the temporal description of the task (i.e., the ordering of its sub-tasks) and the purpose

of the task (i.e., sub- and super-task relationships). The order of sub-tasks is described in terms of task combiners: a sequential combiner that denotes a linear ordering, a parallel combiner that specifies an unordered set of tasks to be achieved, an exclusive-disjunction combiner that specifies a set of tasks of which one must be achieved, a loop combiner, and a conditional combiner. The data flow of a task represents the variables relevant to the task (i.e., its inputs and outputs), and any exogenous computational procedures that map inputs to outputs.

```
name: SampleTask-1
isa: SampleTask
Description: "The user will extract a sample of something and
store it in a container."
subtasks: ()
subTaskOrder: ()
supertask: SampleTestInterpret-1
role: hasSampleTask
SampleContainer: TestTube-1
Attribute: genotype
Sample: blood
interfaceMethod: ClickTransport-1
SampleSource: Laron
SampleDevice: Syringe-1
```

Figure 2. The sample task

Figure 2 shows the frame representation of the **sample** task. This task involves extracting a sample of something and storing it in a container. It has no sub-tasks, and four parameters: the source of the sample, the sample itself, the storage container, and a sample extraction device. Constraints on the bindings of these variables are stored with the slots of the frame. For example, the sample container must be some kind of "container object." MODEST provides a small ontology of types of entities that have distinct states and participate in simple protocols. A container, for example, can be empty or full of some specified content.

```
name: ClickTransport
isa: Transport
Description: "Transport a substance by clicking over a source,
then clicking over a destination"
parameters: (FromContainer, ToContainer, TransportContainer)
states: (before, during, after)
currentState: before
transitions: (
  transfer(FromContainer)   : before-> during
         actions = (extractContent),
  transfer(ToContainer) : during -> after
         actions = (storeContent))
```

Figure 3. The Click Transport scene manager

Figure 3 shows the representation of click-transport, a scene manager for transporting a liquid substance by replacing the mouse pointer with a sample extraction device, clicking over the source to transfer the contents to the extraction device, then clicking over the destination. All scene managers are, essentially, finite state machines. This particular manager defines three states: before, during, and after. When it receives transfer triggers from the object in the appropriate parameter slot (e.g., the "FromContainer"), it transitions to the next state (e.g., "during"), and executes the action associated with the transition (in this case, "extractContent").

```
name: ContainerObj
isa: DiscreteThing
parameters: (content, initialContent)
states: (empty, full)
currentState: empty
content: empty
transitions: (
  mouseDown : empty -> empty, output: transfer,
  mouseDown : full -> full, output: transfer,
  load : empty -> full actions = (loadContainer),
  unload : full -> empty actions = (unloadContainer))
```

Figure 4. The Container Object

Figure 4 shows the representation of ContainerObj, the common abstraction of all containers such as test tubes, syringes, and in the Sickle Cell example, the sample source Laron himself. A container can be empty or full of its specified contents. When the user clicks on a container, it sends a transfer message to its current scene manager. The scene manager is responsible for sending load or unload messages back to the container object.

6. The Design Tool

In this section, we describe the structure and rationale behind MODEST, the tool for parameterizing task models and developing interfaces. Figure 5 shows a screen capture of the tool in operation. On the left half of the screen is the Graphical Model Editor. This displays the current task under refinement and permits the designer to refine a task by clicking on its button. On the right side of the screen is the Dialog Manager. This guides the design process by asking questions in the text box at the top, and allows the designer to select responses from the list box or type in answers in the text field below it. The designer can request examples of unfamiliar concepts by clicking on the Example button. When an interface is fully defined (except for graphical layout), the Preview button is enabled, which takes constructs the actual interface and permits the design to arrange the graphical objects (in author mode) and to dynamically interact with it (in run mode).

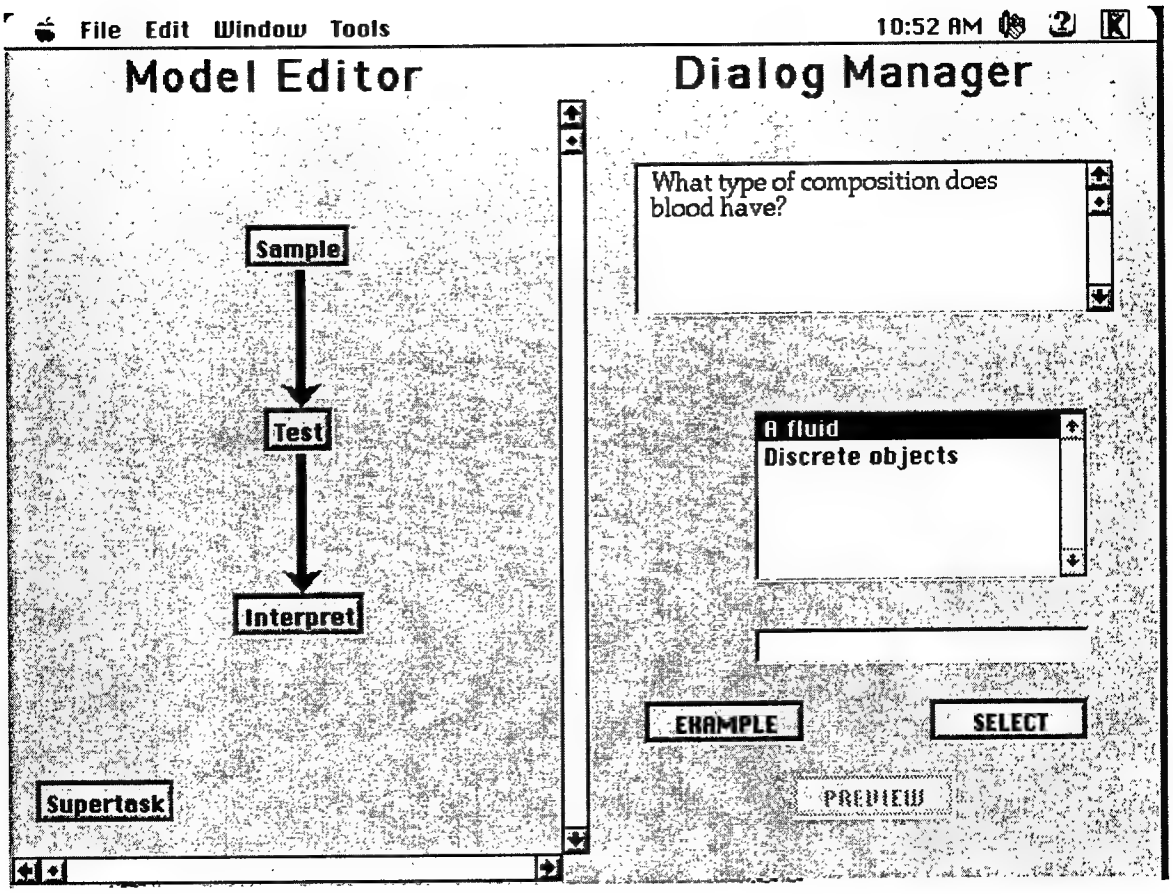

Figure 5. The MODEST Interface

6.1 The Dialog Manager

MODEST is capable of, though not necessarily required to, distinguish between conceptual level design and interface level design. The design process is guided by a question and answer dialog that prompts for design decisions. The dialog is data-driven, and plans for the dialog are hung off of the tasks they refine. The dialog may be structured to proceed in two phases, which correspond for the most part to the conceptual phase and the interface phase. In practice this distinction is not always so clear cut.

6.1.1 Design Plans

Guiding the design of an interface from an abstract root task requires a plan that is distinct from the task model itself. The design plan is essentially a tree of possible questions and branch points. There are a small number of types of design questions that are implemented as distinct classes with different behaviors:

Binding Questions request values for slots.

Specialization Questions permit the designer to specialize a concept.

Decomposition Questions ask the designer to choose a subtask to be refined.

Yes/No Questions provide control over branching in the dialog.

Loop Plans allow the dialog to iterate over a list, subtasks, instances of a concept, or count down.

Primitive Actions automatically create an instance of a concept without asking a question.

Constructing design plans for a given task model can be as complex as constructing the task model to begin with. Depending on the nature of the task, the dialog can be extremely awkward and tedious. Moreover, the dialogs are unforgiving. In the current implementation, there is no easy way to back up and change an answer if you make a mistake.

Because of this, we have come to believe that it would be better to replace some dialogs with specialized data-entry tools that allow form filling when appropriate. Rather than mere frame-editing tools, we suspect that tools for rapidly constructing mappings, assigning graphical representations, etc, would be more appropriate.

6.2 The Graphical Model Editor

The Graphical Model Editor is, in its current form, simply a graphical depiction of the current state of a portion of the plan. It presents a hierarchical task in a flowchart format.

6.3 The Compiler

The compiler takes an instantiated and parameterized task model and constructs an executable interface by instantiating finite state machines to implement the control and data flows and instantiating the presenters to represent the conceptual entities on-screen. The compiler walks through the task model and creates a number of data structures and linkages:

- It constructs the hierarchy of state machine combiners from the declarative task ordering constraints on the model. This entails building the state transitions, because there can be any number of subtasks.
- It performs a numer of simple optimizations based on the task structure. For example, if a task has only one subtask, it drops the subtask combiner and promotes the subtask (*i.e.*, no parallel or serial combinations of singleton tasks). If a task's subtasks were all omitted, it eliminates the combiner and subtasks.
- It creates scenes populated with actors.
- It sets up the presentation hierarchy to mirror the semantic partonomy (*e.g.*, the presenter for a machine contains the presenters for its indicators and controls).
- It creates "begin task" and "exit scene" button and control mechanisms when indicated by boolean flags on the task model.
- It creates the help palette containing the What's Next, Why and How buttons.
- It performs the initial screen layout to ensure that graphical actors won't be buried underneath each other.

- It adds event handler to allow the designer to enter author mode or run mode (by holding down the option key).

6.4 The Graphical Interface Editor

The Graphical Interface Editor is a direct-manipulation WYSIWYG tool for laying out interface elements on the screen. It supports an authoring mode, in which the designer clicks on elements to select them and then can drag them to position or drag their halo to resize them. If the designer holds the mouse button down, a popup menu appears to support other actions, such as adding a text label or change the color.

6.4.1 Layout

The compiler initially lays out the actors in a scene in straight rows so that they are not overlapping or buried. It ensures that sub-presenters are graphically contained within their parent presenters and are in front of them. Actors can be moved by selecting them and dragging them to their desired location.

6.4.2 Resizing

Dragging a corner of the halo resizes the graphical presentation of an actor by scaling its transformation matrix. The aspect ratio is maintained.

6.4.3 Coloring

The "Set color" menu item in the popup menu allows the designer to select a color from a palette to serve as the background of the stage, or as the color of an actor if one is selected.

7. Issues

The viability of our approach depends on addressing two key challenges: generativity and appropriate abstraction.

Flexibility in supporting the design of interfaces for a wide range of tasks entails the ability to compose more complex, aggregate tasks from simpler ones. Achieving this sort of generativity is one of the chief design goals for our task modeling language. The temptation here is to try to provide a fine-grained, fully compositional programming language for task modeling. Even if achievable, however, this would not necessarily give designers the leverage we seek; in the worst case, task model-based design would simply reduce to programming once again.

The alternative we are pursuing is to develop hierarchical libraries of reusable, standardized sub-tasks that can be connected together like circuit boards on the back-plane of a computer,

with the back-plane corresponding to an overarching task such as Sample-Test-Interpret. This simplifies the job of the application designer, while maintaining some degree of flexibility. The design of the sub-task models themselves, like the design of circuit boards, is a more specialized job in our approach.

An approach based on libraries of standardized task models in this way can only succeed if the task models are broadly applicable enough to make their reuse feasible. In other words, they must embody high-level characterizations of user tasks, rather than idiosyncratic details of particular instances of tasks in particular domains. On the other hand, the models must be specific enough to supply operational constraints on interface design. The appropriate level of abstraction for task models is thus determined by a tradeoff between the breadth of their applicability and the specificity of the constraints they provide about the functional behavior of the entities associated with them. The long-term feasibility of our approach depends upon being able to find a rich set of models that strike the appropriate balance.

It is an empirical question whether it is possible to characterize real-world tasks in a way that strikes this balance. In essence, we are banking on the existence of a basic level for tasks, similar to Rosch's basic level for object categories (Rosch *et al.*, 1976). In brief, Rosch and her colleagues have demonstrated that there exist privileged concepts in object hierarchies-e.g., "chair," as opposed to "kitchen chair" or "furniture"-that appear to optimize the trade-off between, roughly, the size of the category and the number of "useful" things you can say about members of the category.

There are three reasons to be optimistic about the existence of basic level tasks as required by our approach. The first is simply the existence of basic level objects, which gives some grounds for extrapolation to other category types. The second is the growing body of evidence that people possess and use a great deal of abstract knowledge about tasks in understanding, planning, and learning, and that such knowledge can be usefully represented and employed by computer programs as well. A great deal of research in AI and cognitive science has established the existence and utility of abstract knowledge of intellectual tasks, such as planning and diagnosis. This work has resulted in the codification of a wide range of models, including Sussman's and Sacerdoti's planning critics (Sussman, 1975; Sacerdoti, 1977), Wilensky's meta-plans (Wilensky, 1982), Schank's thematic organization points (Schank, 1982; Hammond, 1989; Birnbaum & Collins, 1993), Chandrasekaran's generic tasks (Chandrasekaran, 1983), and the KADS models of the Esprit project (Wielinga, Schreiber and Breuker, 1992).

Finally, there is the anecdotal evidence we have collected in the course of analyzing tasks. The Sample-Test-Interpret model described briefly above appears to have wide utility, and we

have in fact instantiated a number of variants leading to different interfaces that support superficially distinct tasks.

To take a more complicated example, consider the task of detecting a threat. To make this concrete, imagine that you are a night watchman making rounds in a building. Suppose that it takes you one hour to make your rounds. Suppose further that, one day, when you return to a location in the building, you discover that in the time intervening since your last visit, something has been stolen from that location. What might you do to prevent this from happening in the future?

In answer to this question, many people respond, "Make your rounds faster." Whether or not this is the solution they propose, everyone will agree that it is reasonable. The question is, what basis do people have for proposing or assessing this solution? No one we have queried has ever actually worked as a night watchman. We must assume, therefore, that it is some knowledge about the task of threat detection in general that permits people to reason sensibly about this situation.

What people need to understand to answer the night watchman question is something like the following model of the situation: Threats must be detected early enough to do something about them. However, it is often too expensive to focus threat detectors in all areas where threats might arise at all times that they might arise. One solution is to schedule threat detection in different locales at different times. However, in order to guarantee that threats will be detected in time, the time course of this schedule must be shorter than the time it takes for threats to come to fruition. Thus tweaking this parameter is one way to address failures to detect threats in time. The point we wish to make here is that we all possess a model of threat detection which is both highly abstract and tremendously detailed. It is exactly models of this sort that we are attempting to represent in our system. In the next section, we describe the refinement and implementation of our model of threat detection.

Part I-b: Threat Detection

8. Introduction

Detecting threats is a common and important task that can greatly benefit from computer support through careful interface design. Intuitively, threat detection seems like a simple task: monitor the world for signs of a threat, confirm the threat and respond to it. In practice, however, it turns out to be a complex task with a rich body of literature devoted to it (*e.g.*, Green, 1986; Williams, 1987). We chose to model and implement interfaces for threat detection for several reasons: it is very different from the sample-test-interpret task, it pushes the limits of what we believe can be done in the task-driven paradigm, and it is a task of vital practical interest.

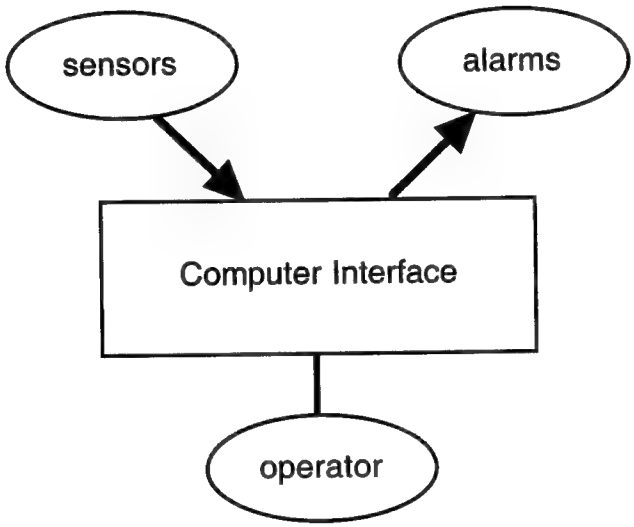

Figure 6. Role of the Threat Detection Interface.

A computer interface for threat detection typically presents sensor data (visually and/or audibly) and permits the operator to raise alarms or operate controls (Figure 6). By embedding an explicit model of the threat detection task in the interface, it can assist in the process of identifying, disambiguating, and responding to threats. The nature of this assistance can be tailored to the types of threats to be detected, the types of sensors available, and the types and numbers of indicators for those sensors.

For example, if a threat has a high cost of false negatives, such as detecting incoming missles, a task-driven interface might require the operator to acknowledge or disconfirm any detected anomalies and might provide escalating levels of alarms if action is not taken promptly.

A night watchman, on the other hand monitors a much less frantic situation. In this case, any activity is abnormal. Some of these anomalies can be detected automatically, through smoke detectors and door sensors. Other activity can be observed through the ubiquitous security cameras. Because this is a relatively simple domain for threat detection, we chose to develop an interface for a night watchman or security guard who must monitor a facility for intruders and fire.

9. The Threat Detection Model

Typically, a night watchman makes his rounds throughout a building checking for suspicious activity. To ensure that he thoroughly and consistently checks the property, there are often time clocks placed along his route where he must register using a special key. If he discovers anything suspicious, such as smoke or a door propped open, he must investigate further until he either confirms a threat or convinces himself otherwise. When a threat is discovered, he will take action, either directly (*e.g.,* by putting out a small fire) or indirectly (*e.g.*, by calling the fire department or police).

Today, technology is available support this process. Smoke detectors, door alarms, and video cameras all aid the watchman by allowing him to montor for threats remotely from a central station. Nevertheless, the basic strategies of detecting threats remain the same. It is still necessary to make rounds (albeit *virtual* rounds), by scanning through closed-circuit video channels. When suspicious situations arise, it is still necessary to confirm or disconfirm threats. The night watchman must still respond to threats when they are confirmed. Because of the ubiquitous and consistent nature of this task, we model threat detection as a basic-level task.

We can summarize this task as an event-driven activity with three main states: **monitoring** for abnormal situations, **classifying** the anomalies as threats or non-threats, and **responding** to threats. This general model is shown in Figure 7.

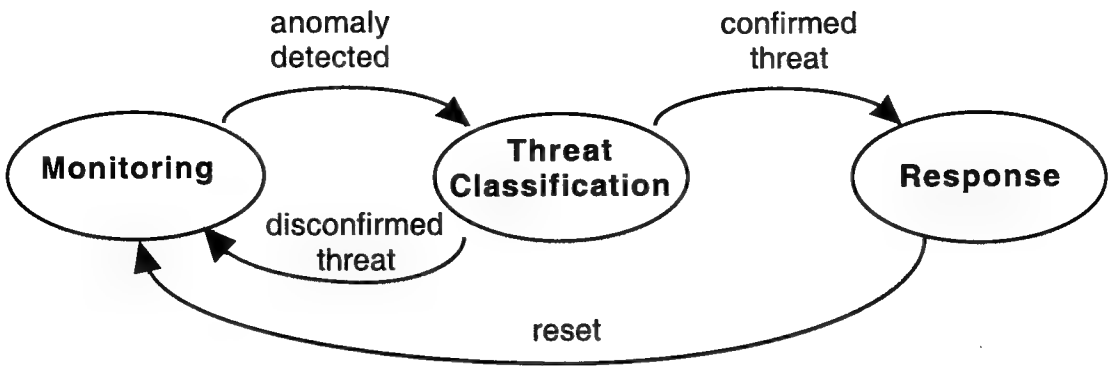

Figure 7. Generalized Threat Detection

While this figure illustrates the common structure of any threat detection task, it over-simplifies the nature of what a night watchman actually does. First, these subasks are not mutually exclusive. The watchman (or security system operator) may need to continue montoring for additional threats, even as he is classifying or responding to a known threat. Second, in a man-machine system such as a high-tech security system, some of the monitoring and classification may be performed automatically for some threats. Third, the generalized model doesn't reflect the wide variation in how these subtasks can be carried out. Further refinement of this model depends on domain-specific aspects of the task (*e.g.*, night watchman v.s. ship-board radar operator) and on the situation-specific resources that are available (*e.g.,* smoke detectors, door sensors, video monitors, etc.)

9.1 Monitoring

As mentioned previously, one way a night watchman monitors for threats is to make rounds, that is, to systematically scan a property for suspicious activity. This is appropriate whenever threats are physically manifested against a spatially distributed facility and when there are limited resources available for monitoring. For some types of threat detection, this makes no sense except in a metaphorical way: *e.g.,* an IRS audit monitors for fraud; an EKG machine monitors for cardiological irregularities in a human patient. However, because security threats are spatially localized, there can be specialized strategies for detecting them. For example, if the primary threat is from intruders, then it may make sense to concentrate security resources at the perimeter of the facility to detect intrusion early. If the primay threat is from fire, then it may make sense to locate smoke detectors where a fire is most likely to break out (*e.g.*, a kitchen) or where it is likely to do the most harm (*e.g.*, a bedroom).

In addition to determining the allocation and placement of sensors, threat characteristics can influence the dynamic strategies for monitoring. For example, if evidence of a break-in is

discovered (such as a broken window or open door), one way to help locate intruders might be to increase the sensitivity of sensors in the facility (*i.e.*, go to "yellow alert"). Because sensors are not always fully reliable, this means accepting a higher rate of false alarms. This is usually a reasonable tradeoff because the conditional probability of the alarm being accurate is greater given the prior evidence.

With current technology, monitoring is facilitated by sensors that operate continuously and autonomously, such as smoke detectors or door sensors. Because they interrupt the operator when they detect an anomaly, these sensors shift the division of labor and change the task from a continuous process to a more intermittent activity. One might be tempted to relegate all monitoring to the automatic sensors and just respond whenever they go off. Sometimes this is appropriate, such as in a hospital, where fire alarms automatically call the fire department. More often, however, it better to keep a human in the loop. Sensors can fail, and moreover these sensors generally don't detect threats directly, but rather they detect *syptoms* of threats, such as smoke levels, temperature, movement, or open doors. Whenever there is a cost associated with false positives, the threat detection task usually includes a classification step to verify or disconfirm a threat.

9.2 Threat Classification

Threat classification is the process of taking possibly ambiguous and spurious sensor data (*i.e.*, the observable symptoms) and determining whether or not there is a threat, identifying what kind of threat it is, and ascertaining its severity. This is a lot like diagnosis, and we've modeled it as a kind of *hypothetico-dedictive reasoning.*

In its fullest generality, this means predicting possible threats from the symptoms and then verifying or disconfirming the threats based on more central or confirmatory features. Predicting possible threats is a subtask we call **hypothesis generation**. Sometimes this is trivial: for example, smoke detectors are designed to detect fire. Other times, this can be arbitrarily hard: for example, it can be quite difficult to determine if an aircraft is a friend or foe based solely on its radar cross section. An interface might support this task by allowing the operator to visually compare his sensor data to a library of images of known threat types. The model we have implemented doesn't include this. Instead, threat types are assumed to be determined either directly by the sensor types or manually entered by the operator. In other words, if the operator sees a fire or intruder on the video monitor, he must tell the computer.

When a potential threat is identified, it must then be verified. The **verification** task can be specialized in a number of ways: One method is to increase certainty by observing the threat

through a redundant sensor of the same type (*e.g.*, see if another smoke detector goes off). Another method might be to integrate sensor data over time (*e.g.*, determine if a high temperature reading is just a spike or if the temperature is really rising). In the night watchman scenario, we specialize the **verification** task to **different-type-sensor-verification**, in which threats detected by automated sensors are confirmed or disconfirmed through manual sensors, in this case video cameras.

One consequence of this verification method is that manually detected threats are implicitly verified. E.g., if a watchman sees fire on the video monitor, he is justified in calling the fire department. In other situations with a higher cost of false positives (such as launching missiles), a formal verification method might be required, even if that merely consists of getting a second opinion.

9.3 Responding

In general, there are many ways to respond to a threat. A fighter pilot may take evasive action; operators in a nuclear power plant may open or close valves or shut down a reactor; a military superpower might launch missiles. In any case, the interface for detecting threats should also, when feasible, support responding to those threats. Often, this means simply communicating the threat to someone else, either a higher ranking decision maker or someone with direct responsibility for taking action, such as the fire department or police.

For a night watchman's security station, responding means confirming or cancelling a threat. At first glance this may seem confusing, because the process of *deciding* whether to confirm or disconfirm a threat is what **verification** is for. **Confirm** and **cancel** are the tasks of telling the computer (and by proxy, the fire department or police) what the operator has decided. One additional complication arises because there may be more than one threat at a time. Consequently, in order to confirm or cancel a threat, the operator must first designate the threat to which he is referring.

This raises interesting possibilities that we have not yet addressed: If two smoke detectors go off, is it the same fire? When one fire threat is cancelled, should it cancel the other? A more advanced threat detection mechanism might look for relationships between proposed theats and try to help the operator to see emerging patterns.

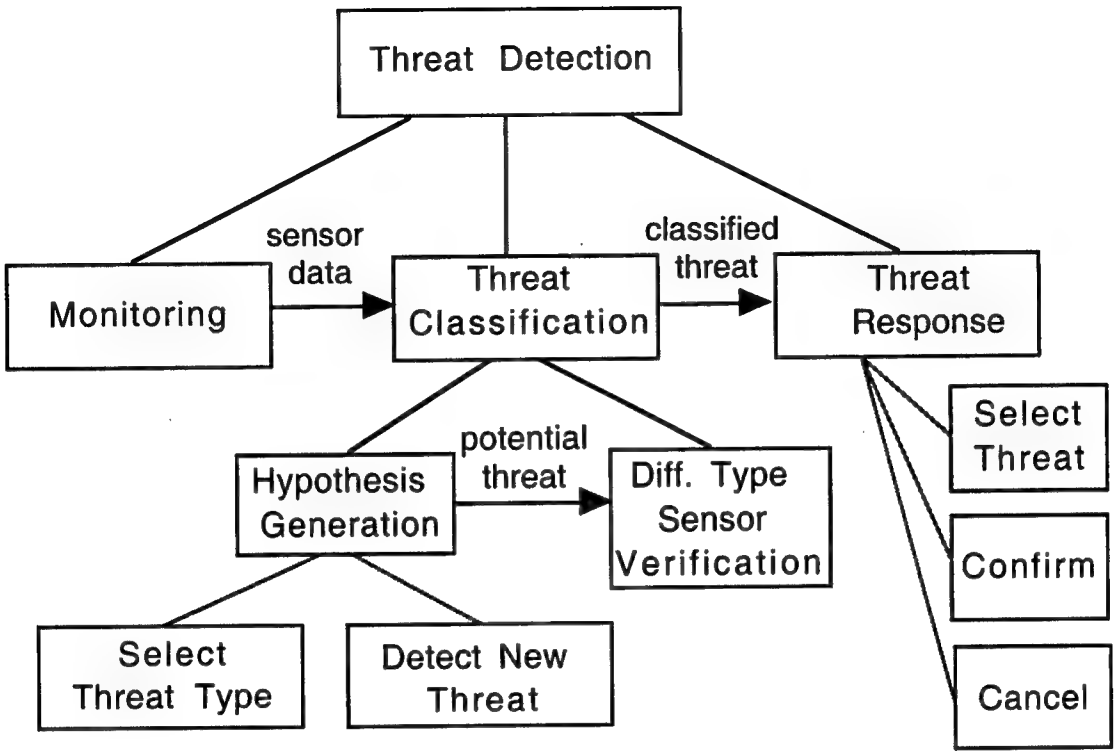

Figure 8. Elaborated Threat Detection Model.

Figure 8 shows our refined and elaborated model of threat detection for the night watchman domain. It depicts the subtask breakdown and some of the dataflow across subtasks, but it does not indicate the control structures that constrain the operators actions. This is largely because there are very few such constraints. Whereas previous tasks we have modeled have been highly procedural (*e.g.*, Sample-Test-Interpret), Threat Detection is driven by external events. Many of the subtasks are optional, or more accurately, the subtasks are not always shared between the the operator and the system. For example, the system may automatically monitor and propose threat, leaving the operator to verify and confirm them. Alternatively, the operator may detect and verify threats on his own. The only real constraints on what must be done are that potential threats that are generated must be cancelled or confirmed, and monitoring must resume before too much time has elapsed. In the interface we have developed, the video monitor reverts to automatic scanning if it is left in manual too long.

9.4 Auxiliary Models

In addition to the task model itself, there are three types of auxiliary models that are required to fully define the threat detection task:

1) *Sensor & Indicator Models* represent the properties and behaviors of different classes of detectors in the world and their associated indicators and alarms.

2) *Threat Models* represent the types of threats to which the human operator will be monitoring and responding.

3) *Situation Models* are abbreviated representations of the current state of the world with respect to sensors and potential & known threats.

Parameterizing a model of threat detection therefore entails identifying the types of threats to be detected, the sensors and indicators available, and the types of actions the user can take to respond to threats. In the rest of this section, we will describe the different kinds of models and what they represent.

9.4.1 Sensor Models

Sensors can be broadly divided into two types: *manual* and *automatic* . Manual sensors relay information to the operator in a form that is representationally opaque to the computer program, such as a video stream from a camera. Automatic sensors represent their data in computer manipulable form. These are further divided into *linear* sensors, *binary* sensors, and *threshold* sensors. Linear sensors, such as thermometers, have a numerical value but must be polled to read that value. Binary sensors, such as door sensors, send messages when their state changes. Threshold sensors, such as smoke detectors, are hybrids that have an underlying linear sensor with a settable threshold. When the reading exceeds the threshold, they send messages like binary sensors.

In our model, when binary and threshold sensors send messages, the situation manager that recieves the messages constructs objects representing potential threats. These potential threats may vary in their degree of certainty and the specificity of the type of threat they represent, depending on the type of sensor. For example, smoke detectors create fire threats, while temperature sensors or current sensors might construct generic threats.

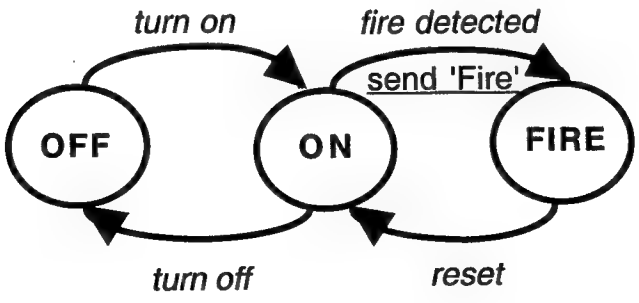

Figure 9. Smoke Detector states.

Figure 9 shows the state transitions for a smoke detector. The protocol for a sensor is the set of triggers in its state transition table (shown in italics) and the actions it can take (underlined). As with all devices, a smoke detector can be disabled, in which case the only message it responds to is the 'turn on' message. The trigger to its threat state is just '<X> detected', where X is the name of the threat type. The action is to send the trigger <X> to the situation manager.

The smoke detector has a linear component that records the carbon-dioxide level. The interface between the linear component and the discrete state machine is a generic mechanism that translates continuous parameters into discrete ranges or thresholds. When the level exceeds the specified threshold, it sends the *fire detected* trigger to the state machine.

9.4.2 Indicator Models

An *indicator* presents sensor data to the operator. When there is more data than can be presented at one time, given constraints on screen real estate, a single indicator may be *multiplexed* so that it can display data from several sensors at different times. For example, a security guard will often have a single video monitor that switches between cameras, rather than a bank of dedicated monitors. A map-based presentation may show only part of the territory under surveillance at one time, such as one floor of a building. When indicators are multiplexed like this, the criteria for verifying a threat becomes slightly more complex.

Specifically, for a user to cancel or confirm a potental threat, he must have, at minimum, observed the relevant data in order to have acquired the features necessary for confirmation. When confirmatory features are observed through a multiplexed indicator such as a video monitor, that indicator must be turned to the right channel to display the appropriate information. In fact, the precondition for verifying such a threat is not observing the monitor, nor is it the act of changing the channel to the right camera, since the monitor may already be pointing to the correct camera. The actual precondition is that the monitor must have been pointing to the appropriate camera sometime after the threat arose and before the threat is confirmed or cancelled. To do this, it is necessary to timestamp threats as they are logged in the audit trail and compare them to the monitor channel changes.

9.4.3 Threat Models

At the center of our model of threat detection is the explicit representation of the concept of a threat. The formulation and manipulation of potential and confirmed threat instances is one of the sources of power of a task-model centered system.

A threat model consists of (at least) the following information:

Type: (e.g., fire, intruder, etc)

Symptoms: (e.g. smoke, heat, door)

Verification method: (e.g., visual observation)

Criticality: (e.g., cost of false positives, cost of false negatives)

Time criticality: how much time to allocate for verifying a detected threat

Corrective Actions: things to do when a threat is confirmed.

Each symptom corresponds to a sensor type that can be used to detect it.

An instance of a threat indicates the type of threat, the time it was detected, the sensor that detected it, the certainty of the threat, and any action taken to resolve it. As threats are created and resolved, they are added to and removed from the situation model.

9.4.4 Situation Models

The situation is a model of the world at a given time. It maintains the following information:

Sensors: a list of all the sensor instances in the system

Threats: a list of current threats or potential threats being tracked

State: the current state of the system. (e.g., no threat, multiple threats, fire detected, etc.)

Task Agenda: a list of all the tasks currently being pursued

Because the situation model supports multiple tasks being pursued simultaneously, we model it as a subclass of the concurrent task combiner.

10. Developing a Threat Detection interface

In this section, we present an example of the design process using MODEST to construct a threat detection interface for a night watchman. This process proceeds from the abstract properties of the anticipated threats and progressively constrains the program. As with Sample-Test-Interpret, the design process entails three main phases: conceptual design, interface-level design and graphical layout. The first two phases are guided by the design dialog, while graphical layout is performed through a direct manipulation editor.

10.1 Conceptual Design Phase

The dialog begins with some general questions about the number and nature of threats that are to be detected:

Choose a subtask to implement:

-> **Threat Detection**

How many types of threats are you detecting?

2

What is the name of one type of threat you are detecting?

fire

What is the name of another type of threat you are detecting?

intruder

The system does not initially have representations of fire or intruders, so it asks questions about what types of resources are available for detecting these types of threats in this particular situation:

How many types of sensors are used to detect fire?

2

What is a type of sensor used to detect fire?

-> **video Camera**

What is another type of sensor used to detect fire?

-> **smoke Detector**

The concept of smoke detector does exist, but it is represented simply as a specialization of a threshold detector. The dialog now proceeds to ask how a fire threat should be indicated to the operator:

How many alarms are used when fire is detected?

1

What is the name of an alarm used when fire is detected?

fireAlarm

What type of alarm is fire?

visual Alarm

The design tool now proceeds to ask about sensors and alarms for detecting intruders:

```
How many types of sensors are used to detect intruder?
2

What is a type of sensor used to detect intruder?
->   video Camera

What is another type of sensor used to detect intruder?
->   door sensor

How many alarms are used when intruder is detected?
1

What is the name of an alarm used when intruder is detected?
intruderAlarm

What type of alarm is intruder?
audio Alarm
```

A common way to present the state of the world to an operator monitoring for threats is to display the state of the sensors on a map. Instead of reasoning about whether this is appropriate, the current system simply asks:

```
Do you want to present the sensors on a map?
->   Yes
```

Now that the system has represented the types of resources available for detecting threats, it proceeds to guide the user in refining the monitoring subtask of threat detection:

```
Choose a subtask of Threat-Detection to refine:
->   Threat Monitoring
```

At this point, it is necessary to instantiate the specific sensors that will be used in the monitoring task:

How many videoCameras are there?

2

What is the name of a videoCamera?

c1

What is the name of another videoCamera?

c2

The two video cameras could display their outputs either on two separate monitors or on one multiplexed monitor. Due to limited screen real estate, we choose a single monitor:

How many indicators does a videoCamera have?

1

What is the name of an indicator of a videoCamera?

videoMonitor

Is videoMonitor an indicator shared by all videoCameras?

-> yes

What type of shared indicator is videoMonitor?

-> Multi Input Video Monitor

The design tool now proceeds to instantiate the second type of sensor for detecting fire threats:

How many smokeDetectors are there?

2

What is the name of a smokeDetector?

s1

What is the name of another smokeDetector?

s2

Because smoke detectors are threshold sensors, they have a linear reading as well as a binary state. We choose to display this reading next to the icon for the smoke detector:

```
How many indicators does a smokeDetector have?
1

What is the name of an indicator of a smokeDetector?
CO2level

Is CO2level an indicator shared by all smokeDetectors?
->   No

What type of single indicator is CO2level?
->   Digital Readout
```

The design tool now needs to instantiate sensors for detecting intruders. Since we have already described the video cameras, it only needs to ask about door sensors:

```
How many doorSensors are there?
2

What is the name of a doorSensor?
d1

What is the name of another doorSensor?
d2
```

A door sensor is a binary device, so there is no need for an indicator beyond its icon on the active map. At this point, the monitoring sub-task has been defined at the conceptual level and the dialog moves on to refine the threat classification sub-task.

```
Choose a subtask of Threat-Detection to refine:
->   Classify
```

A threat is classified using the hypothetico-deductive method: predict or hypothesize a type of threat given anomalous sensor data, and confirm or reject that hypothesis. In this system,

hypothesis generation is performed automatically by some types of sensors, but it is necessary whenever the operator observes a threat through the video monitor or directly.

```
Choose a subtask of Threat Classification to refine:
->    Hypothesis Generation
```

Hypothesis generation can be arbitrarily complex (as it often is, for example, in medical domains). Here, the operator will simply pick a threat type from a listbox and create an instance of that type, if necessary.

```
Choose a subtask of Hypothesis Generation to refine:
->    Select a Type of Threat

Choose a subtask of Hypothesis Generation to refine:
->    Detect a New Threat

How should the user detect a new threat?
->    Press a button
```

Once there is a potential threat posted, the next step is to confirm or cancel the threat:

```
Choose a subtask of Threat Classification to refine:
->    Verify
```

Given a symptom of a threat (i.e., anomalous sensor data), it could conceivably be verified in a variety of ways. Does it persist over time? Can it be confirmed through redundant sensors of the same type? In this interface, we choose to use the video cameras to verify threats detected by the automatic sensors. This keeps the human in the loop, which is probably a good design decision.

```
What type of verification will the user perform?
->    Different Type Sensor Verification

What type of sensor is going to verify a fire detected by a smokeDetector?
->    video Camera
```

When using one sensor to verify another, it is critical to ensure that both sensors cover the same region. This is the main reason why the sensors were instantiated so early in the

design process. If a graphical tool were available to connect primary and verification sensors, this would not be necessary and sensors would not need to have names.

```
What sensor is going to verify s2?
->   c2

What sensor is going to verify s1?
->   c1

What type of sensor is going to verify a intruder detected by a doorSensor?
->   video Camera

What sensor is going to verify d2?
->   c2

What sensor is going to verify d1?
->   c1
```

The last major subtask of threat detection is responding to a threat.

```
Choose a subtask of Threat-Detection to refine:
->   Respond to Threat
```

Since there may be multiple threats simultaneously, the first step in responding to a threat is to select the threat to which the operator will respond:

```
Choose a subtask of Threat Confirmation/Cancellation to refine:
->   Select a Threat
```

The tool instantiates the select-a-threat subtask and will later ask for the specific method for acheiving it (e.g., from a popup menu or from a listbox).

```
Choose a subtask of Threat Confirmation/Cancellation to refine:
->   Cancel a Threat

How should the user cancel an existing threat?
->   Press a button
```

```
Choose a subtask of Threat Confirmation/Cancellation to refine:
->    Confirm a Threat

How should the user confirm an existing threat?
->    Press a button
```

10.2 Interface Design Phase

Now the conceptual phase of the design is complete and the design tool requests specific graphic and sound resources to define the look and feel of the interface.

```
What does the map look like?
->    smallFloorPlan.PICT
```

Earlier, we elected to indicate a fire threat by means of a visible alarm. Now we have to provide the graphics to show what the alarm looks like in its different states:

```
What does fireAlarm look like when it is off?
->    AlarmNoFire.PICT

What does fireAlarm look like when it is on?
->    AlarmFire.PICT

What is the sound for intruderAlarm?
->    CHIMES.WAV
```

Each leaf task in the model is achieved via a specific user-interface action. Monitoring is implemented by the scanning method which updates the display based on user-initiated events, such as selecting a camera on the video monitor, and external events, such as smoke detectors going off. It is unclear in retrospect whether scanning should be an interface action or a specialization of monitoring.

```
What type of screen action should the user perform to monitor the sensors?
->    scanning
```

Sensors are depicted as icons on the map. A sensor has a different icon for each of its states:

What does c1 look like when it is off?

-> **videoCameraOFF.PICT**

What does c1 look like when it is on?

-> **videoCameraON.PICT**

Although camera2 looks identical to camera1, this need not be the case (and isn't for the door sensors), so the design tool asks for icons for each instance of each sensor.

What does c2 look like when it is off?

-> **videoCameraOFF.PICT**

What does c2 look like when it is on?

-> **videoCameraON.PICT**

What does s1 look like when it is off?

-> **SmokeDetectorOFF.PICT**

What does s1 look like when it is on?

-> **SmokeDetectorON.PICT**

What does s1 look like when it is Alarm?

-> **SmokeDetectorFIRE.PICT**

The operating range of threshold sensors is between 0 and 100. When their values exceed that threshold, they trigger their alarm.

What is the current value of the data measured by s1?

45

What does s2 look like when it is off?

-> **SmokeDetectorOFF.PICT**

What does s2 look like when it is on?

-> **SmokeDetectorON.PICT**

```
What does s2 look like when it is Alarm?
->    SmokeDetectorFIRE.PICT

What is the current value of the data measured by s2?
78

What does d1 look like when it is off?
->    northDoorOFF.PICT

What does d1 look like when it is on?
->    northDoorCLOSE.PICT

What does d1 look like when it is alarm?
->    northDoorOPEN.PICT
--
What does d2 look like when it is off?
->    southDoorOFF.PICT

What does d2 look like when it is on?
->    southDoorCLOSE.PICT

What does d2 look like when it is alarm?
->    southDoorOPEN.PICT
```

We now select the interface actions to implement the remaining leaf tasks:

```
What screen action should the user perform to select a Type of threat?
->    Select an item from a listbox

What type of screen action should the user perform to verify a threat?
->    Select an Information Channel
```

SelectInformationChannel displays the appropriate camera view when the operator changes channels on the video monitor. This is actually a complex process, because it also keeps track of the last time a channel was viewed. This allows the system to ascertain whether or not the monitor has displayed the appropriate channel since the time a threat was posted.

```
What type of screen action should the user perform to select a threat?
->   Select an item from a listbox

Press Preview to display the interface.
```

At this point, the graphical objects are laid out on the screen, roughly in a grid. The designer arranges the objects by selecting and dragging them to their desired locations and resizing them if necessary.

10.3 The Night Watchman's Interface

Figure 10 shows the final threat detection interface for a night watchman. The interface supports fire and intruder detection using smoke detectors, door sensors, and closed circuit television. In Figure 10, the interface is being driven by an external simulation that sends events to the sensors and provides a video stream from pre-recorded movies.

The situation is displayed on the *active map*, in the top left. This shows the floor plan of the building, the locations of all the sensors, and their current state. A single video monitor can be directed to display the video feed from either camera. Cameras can be selected by clicking on them in the map or by cycling through the previous or next buttons on the monitor. A single visual alarm shows the global state of the situation (in this case, indicating a fire). The scrolling window below displays the log that automatically records and timestamps threats as they happen. The selection box below the log permits the operator to select a type of threat and manually construct and record the threat if, for example, he observes it on the video monitor. Potential threats that are detected by smoke detectors or door sensors can be confirmed or cancelled through the buttons in the middle. The help palette at the far right provides instructions on performing the task.

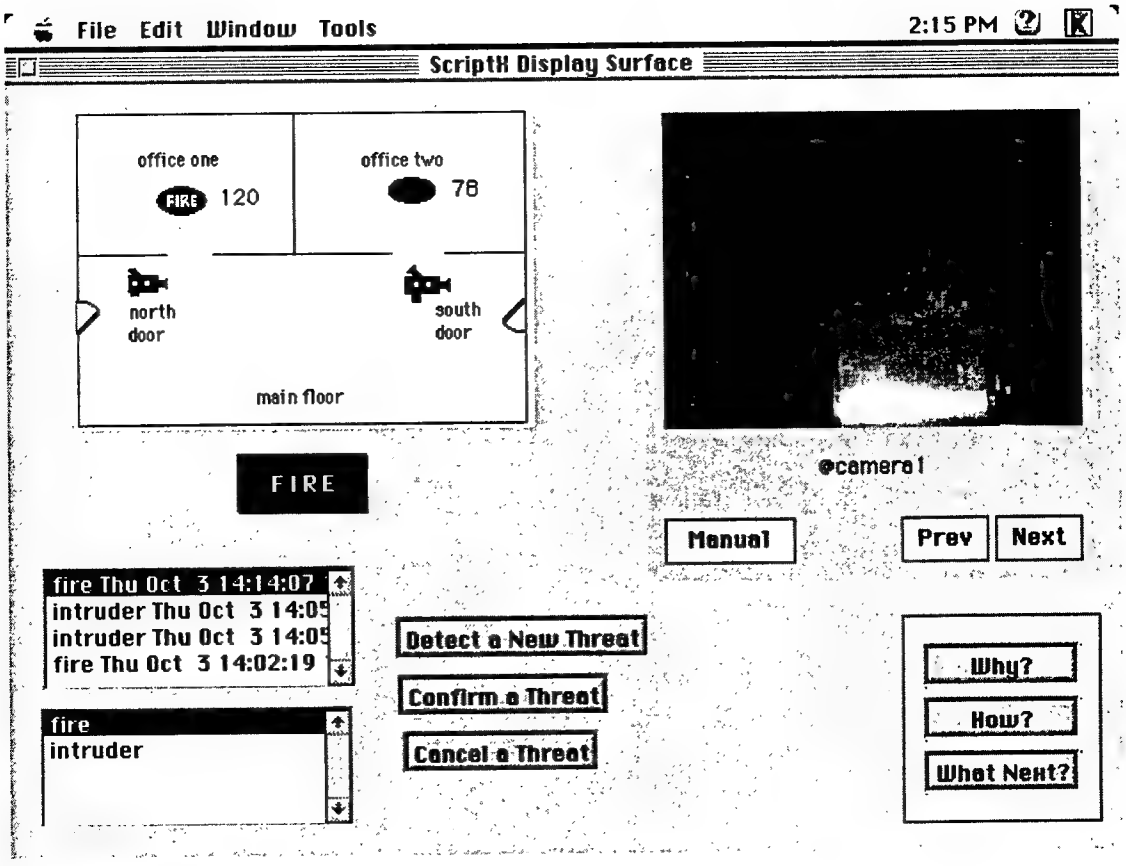

Figure 10. Threat Detection Interface.

The operator monitors sensors in the active map and selects cameras to view in the video monitor. When a smoke detector exceeds its threshold or a door sensor is activated, the system logs a potential threat and the user must then view the area with the appropriate camera and either confirm or cancel the threat. Alternatively, if the user detects a threat through the video monitor, he can manually log the threat by clicking on the "Detect a New Threat" button and selecting the type of threat detected. In a real security system, this would set off the appropriate alarms and call either the fire department or police as required.

11. Implementing the model

In order to implement the threat detection model, it was necessary to implement new mechanisms to extend the MODEST runtime. The capabilities that were added are not specific to threat detection, but fall under the general headings of 1) surmounting the limitations of finite state machines, 2) permitting multiple simultaneous tasks, 3) supporting runtime instantiation, task dispatching, and external events.

11.1 Surmounting Limitations of State Machines

One of the issues in this research has been finding an appropriate tradeoff between simplicity of the design process and flexibility of the resulting software. In particular, the use of state machines as an underlying mechanism has benefits and drawbacks. In so far as they encapsulate behavior behind a simple protocol of messages, state machines are compositional and reusable, and the state machine model is valuable. Nevertheless, there are many occasions when a pure finite state machine implementation is inadequate. We employ three main mechanisms to surmount the limitations of state machines: 1) State transitions can execute complex functions rather than simply output other triggers, 2) Triggers carry parameters with them that can determine whether they are accepted or not and provide arguments to the action functions on transitions, and 3) The parent state machine managers to which graphical actors send their messages changes dynamically as the user proceeds through his task. This enables the actors to serve different tasks at different times, such as the test-tube object that is shared between the **sample** and **test** tasks in the Sickle Cell demo example. We believe these extensions to the finite state machine mechanism do not significantly compromise the compositionality and reusability of the objects.

11.2 Permitting Multiple Tasks

The threat detection model assumes that the user may be engaged in multiple tasks simultaneously. The means that the whole idea of using user-interface events to step through the task and enable or disable actions is no longer sufficient. Instead, we implemented a *concurrent* task combiner that doesn't disable alternative subtasks.

This raises the problem of how to handle actors and widgets that are shared by different tasks, such as the video monitor, which may support both monitoring and verification. To whom should they send their messages if there isn't a unique current task at any moment? We addressed this problem by leaving the video monitor reporting to the monitoring task's manager and passing unrecognized messages up the hierarchy until they are recognized. This works only because the top level manager in this case is a *situation manager*, which is a special kind of concurrent combiner that recognizes particular triggers. A more general solution, we believe, is to relax the requirement that actors must have a single manager, and instead allow them to broadcast their messages to a set of managers.

In addition to the concurrent combiner, we also implemented conditional and loop combiners. The conditional combiner is a disjunctive branch that allows one task or another to be taken based on data, rather than on an event. The loop combiner iterates over its subtasks, but this turned out to be unnecessary for threat detection since the concurrent combiner never terminates.

11.3 Instantiation, Task Dispatching & External Events

One of the benefits of a model-driven interface is that it can make explict the conceptual entities with which the user interacts, and in so doing, can simplify the man-machine interaction. Rather than employing an ad-hoc collection of rules that directly relate input data to output presentation (*e.g.*, if sensor value exceeds preset threshold, display it in red), we employ a *model-view-controller* mechanism. One implication of doing this is that we end up instantiating objects at runtime, such as threats. This raises the level of complexity, because tasks have to refer to specific objects that don't exist at design time. For example, the operator must verify *the threat that was detected*, not just some platonic ideal of a threat.

On the whole, however, the benefits outweigh the drawbacks. By maintaining an explicit model of the threat, it is possible to enable, disable, or invoke tasks based on data, rather than solely on events and user actions. Moreover, in an environment in which multiple threats are can occur nearly simultaneously, it becomes possible to distinguish which threats were detected by which sensors, which have been confirmed, and which have been responded to.

Time-stamping threats also permits the recording of an audit trail. In addition, instantiated (potential) threats are the mechanism by which we achieve sensor fusion. The threat records the sensor that was used to detect it and records the time it was first detected. Now to verify the threat, the verification task looks at the sensor that detected the threat (the 'predictive feature') and looks up the sensor that should be used to verify threats detected by that sensor. There is no need to reason about geometry and what's in the field of view of a particular camera, because we indicated at design time which camera could see the area monitored by each smoke detector or door sensor.

11.3.1 Task Tracking in Procedural- and Event-Driven Tasks

In order to provide context-specific help and advice, the system relies on tracking the user through the explicit model of the task. This can be relatively easy for a procedural task because there is what amounts to a 'program counter' which, if not literally expressed, can be inferred from the state of the scene manager finite state machines. For example, to find the next tasks that can be performed, the algorithm traverses the hierarchy of scene managers until it finds the managers that are in their initialized, unlocked, states and looks up the tasks they implement.

This works well for procedural tasks in which the set of actions the user can take is entirely determined by what he has done so far. Here, the state of the interface is determined entirely by user-initiated events. For tasks that are driven by external events, such as threat

detection, this doesn't work. For example, in threat detection, the next task to perform depends on whether there are any open threats on the agenda that must be verified or responded to. We can't infer what to do based on which managers are initialized or locked. In other words, for procedural tasks, tracking is event-driven; for event-driven tasks, tracking is data-driven. We have not implemented data-driven task tracking in MODEST, so the help palette does not work for the threat detection task.

12. Discussion

We have barely scratched the surface in the rich area of threat detection. Probably the biggest shortcoming of what we have done so far is the inability to construct a coherent picture of the overall threat situation. There is no facility for the user to combine multiple threat instances into one. EG, if two smoke detectors go off, there's probably only one fire. Alternatively, if an intruder is detected followed by a fire alarm, then the intruder probably started the fire, possibly as a diversion to cover some other activity. Putting together a coherent story from primitive sensor data is beyond the scope of our approach to task-driven interaction.

Also, more threat-specific stategies for monitoring and responding to threats would be appropriate. For example, monitoring strategies might include scanning for intruders more closely at the perimeter than in the interior of a region. If a breakin is detected, verification typically involves searching for intruders. This search should progressively widen as time goes on, taking into account the speed with which intruders are likely to move. Moreover, it should be possible to adjust the sensitivities of the sensors when trying to verify a particular threat. Sensors that are normally prone to false alarms might have their gain turned up when an anomaly is detected.

One capability that would be useful in an interface for threat-detection would be a parasitic or meta-level task to ensure that the operator is awake and at his station. For example, if cameras are not scanned or alarms are not reset after some period of time, a dialog box should appear and request attention. If no activity is detected, the system might call another operator to investigate. This is analogous to the time clocks located at points along a night watchman's rounds.

Part I-c: Functional Organization of Interface Libraries

13. Introduction

The human-computer interface community has devoted a great deal of effort in recent years to the creation of interface development environments, both academic and commercial. These environments are intended to simplify the process of designing an interface by providing such capabilities as the graphical manipulation of prototypes, and the reuse of standard interface objects stored in libraries.

In this section, we will present an approach to improving these libraries by organizing them around the functionality of the stored interface constructs. This should enable interface designers-particularly novices-to more easily find the best collection of interface objects for performing a given task.

14. Current Organization of Interface Libraries

Currently available interface libraries and their documentation tend to organize their constituents in one of two ways:

- Listed by name or icon (e.g., on a palette), often alphabetically (Cwikla; Scott *et al.* 1995).
- Categorized hierarchically by behavior and appearance (Kaleida, 1995).

Using the first method of organization is essentially the same as using no organization. Interface designers must rely on prior knowledge of the names or appearances of interface objects to find them in the library. Browsing through a large library organized in this way can become frustrating and tedious.

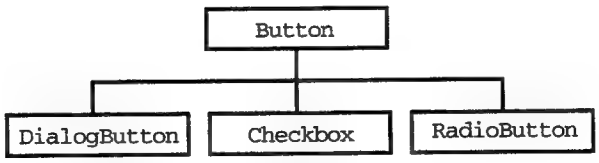

Figure 11. A typical object-oriented organization of interface objects.

The second method of organization directly reflects the object-oriented programming techniques often involved in building an interface library. A typical categorization scheme is illustrated in Figure 11: Dialog buttons, checkboxes, and radio buttons are all stored under the Button class, which allows them to inherit the basic button behaviors and appearance. Organizing interface objects in this manner is efficient for the computer, but not necessarily

for interface designers who are searching for interface objects to incorporate into their design. For instance, a dialog button and a radio button may share the same basic button behaviors, but their functionality and context of use is quite different. A dialog button is used to issue a command during the course of running a program. A radio button, in contrast, is used to choose an item from a set without any presumption that the choice will immediately impact program behavior, and so has more of the flavor of a data entry device.

The main problem with both the simple list and object-oriented organizations of interface libraries, then, is that they don't necessarily correspond to the ways that designers, particularly novices, think about their problems. Designers must first consider functionality-what tasks the users of the system will be performing-and then try to find interface objects that are most appropriate for those tasks. Unfortunately, using today's interface libraries, designers must perform the second step with search keys completely different from the functional specifications derived in the first step. Only after remembering or guessing possible names, behaviors, and/or appearances of interface objects can designers access these objects to determine whether they have the appropriate functionality. For instance, in the interface guidelines for the Macintosh (Apple Computer, 1995), readers must specifically choose to read about radio buttons before they can discover what they are used for. The upshot is that after determining what the user needs to be able to do, designers often end up checking whether interface objects x, y, z, etc., will support the user appropriately.

Basically, then, today's interface libraries are similar to a university library that lacks subject categorizations or the Dewey decimal system. Anyone looking for a book on human-computer interaction would not have the luxury of finding all relevant books listed in one place; instead, they would have to search for titles that mentioned words associated human-computer interaction, authors associated with human-computer interaction, and so forth.

Granted, current interface libraries are small enough that they can, if necessary, be browsed in this way. For novices, however, this can be an extremely difficult process. Moreover, even experienced designers may be led to make inappropriate design decisions in this way. Even though the number of widgets available in an individual design tool may be small, the number of widgets available to an individual designer grows much larger as specialized libraries are added. The problem only gets worse as interface designers accumulate a personal library of widgets and combinations of widgets from sources such as commercial libraries, previous personal designs, and coworkers.

15. Functional Organization of Interface Libraries

In order to solve the inability of designers to search interface libraries in a more natural and intuitive way, we propose to organize interface objects according to their functionality. We want to forge a direct link from the task that designers are thinking about to the interface objects that they should use. Using this approach, designers should need to consider only two questions:

- What do I want the user to be able to do here?
- Which interface objects can support that?

As a simple example, if a designer decides that a user should make an exclusive choice at some point in a task, the designer should be able to go straight to the category "widgets used for making exclusive choices" in a library and find a set of radio buttons, popup menus, and any other construct that enables users to make exclusive choices.

16. Model-Based Interface Design

Our approach to functionally organized interface libraries is part of a larger project on task model-based interface design (Hinrichs *et al.*, 1996). A key focus of the project has been the construction of a design tool, MODEST (MOdel-based Design Employing Standardized Tasks), that uses explicit, standardized task models to drive the interface design process, resulting in the automatic compilation of large portions of the interface. In order to automatically generate interfaces from task models in this way, MODEST needs to know what interface constructs are appropriate for the task models with which it is working. The best way to provide this support is to index these constructs in terms of the tasks for which they are used-in other words, to build a functionally organized interface library.

MODEST's design is based on the assumption that there exist a number of standardized tasks that are general enough to cover a wide variety of specific applications, and yet provide enough specific content to drive the development of useful interfaces. So far, we have studied several tasks that we believe possess these qualities, including the following:

- Parameter Setting
- Visual Comparison
- Sample-Test-Interpret, a general lab experiment task
- Threat Detection
- Strategic Planning
- Iterative Hypothesis Generation and Testing
- Simulation

For example, a task like Sample-Test-Interpret (in which the user takes a sample, runs a test on it, and interprets the results) is general enough to produce many diverse interfaces for jobs like blood testing or water sampling, but specific enough to guide the selection and appropriate parameterization of specific objects for use in these interfaces.

As the list above makes clear, the tasks we are investigating span a range of levels of abstraction. For instance, Parameter Setting and Visual Comparison are both relatively small, low-level tasks that reside at the border between the user's real-world task and the interface constructs used to support that task. On the other hand, Iterative Hypothesis Generation and Testing is a much more complex and abstract task, which might, for example, employ Simulation or Sample-Test- Interpret as subtasks. Building interfaces for a wide range of complex tasks by combining simpler task models and their associated interface constructs in this way-a type of "plug and play" capability for interface design-is another goal of our project.

The current incarnation of MODEST comprises three main phases:

- A dialog in which the designer selects a task model and parameterizes it appropriately for the particular application at hand.
- A dialog in which the designer specifies appropriate interface idioms and graphical objects to represent the actions and conceptual entities specified in the task model.
- An editing session in which the designer previews the generated interface and arranges its components as desired on the screen.

During the first dialog phase, designers select an abstract task model and then answer questions to specify parameters of the model with values that are appropriate to the target task. For instance, if the Sample-Test-Interpret task were being used to generate an interface for a blood test, a designer would answer questions about the sample (i.e., blood), the sample site (i.e., a person), the sample device (i.e., a syringe), and so forth. In the next stage, designers specify the media they want to represent these objects in the interface and the interface interaction paradigms used to represent actions involving these objects. For example, a designer would be asked to select pictures of what the syringe looks like before and after the blood is taken from the person, to select how the blood is transferred from the syringe to a storage container (e.g., by clicking on the container with the syringe), etc.

After this dialog, MODEST generates a working interface from the parameterized task model, at which point the designer can test the interface and adjust the layout and size of the interface objects. Note that the chief focus of MODEST is not the layout of the generated interface, but its functionality. We are most concerned with the objects that should comprise the interface and how those objects interact.

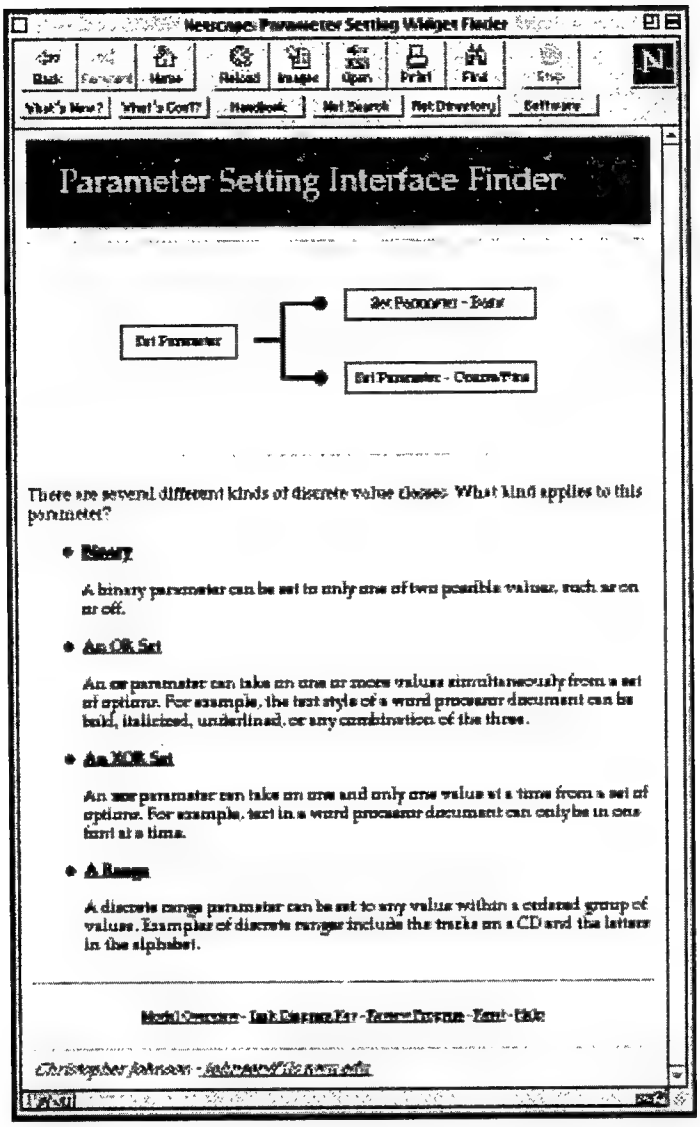

Figure 12. A sample screen from a functionally organized interface library for parameter setting.

17. Parameter Setting

As mentioned above, both parameter setting and visual comparison are relatively small, low-level tasks that, as a consequence, have very high possibilities for reuse. In addition to the modeling these tasks in the MODEST environment, therefore, we have also implemented semantically organized widget libraries for these two tasks in HTML. The resulting systems guide the designer through a task model, asking questions about the context of use, ultimately outputting a list of interface objects that could be used in an interface for the task.

In order to illustrate the structure and use of our functionally organized libraries, we will first summarize our task model of parameter setting, and then step through a typical interaction with our library of interface objects for parameter setting organized around that task model.

17.1 A Task Model of Parameter Setting

Figure 13 shows the top level of our task model for parameter setting. The task can be specialized as one of two types in our model: the basic parameter setting task, or the coarse/fine parameter setting task. The basic parameter setting task (shown in Figure 14) involves alternately checking and adjusting the parameter in a tightly closed loop until the parameter is set to some target value. The coarse/fine parameter setting task (shown in Figure 15) is often used when a parameter's value has a large number of options or must be very exact. The only difference between coarse/fine and basic parameter setting is the option of adjusting the parameter in larger or smaller increments.

These figures describe the basic flow between the subtasks of parameter setting at a relatively course level. Our task model also contains a representation of the objects involved in parameter setting: parameters, displays, and controls. Our criteria for classifying a parameter, depending upon the nature of its range, is shown in Figure 16.

The range of a parameter can be discrete or continuous. If the parameter range is discrete, it can be binary (e.g., an on/off parameter); an OR set, which can have several of its values selected simultaneously (e.g., a word-processing text style); an XOR set, which can have one and only one of its values selected at a time (e.g., the font of a text selection); or a range of ordered values (e.g., the letters of the alphabet). Sets can be further distinguished as containing items that are categorized into subsets. For example, a user may have to specify a selection from a set of geographical locations that are hierarchically categorized (e.g., country, state, county, city).

Next, a parameter's composition can be one-, two-, or three-dimensional (higher dimensions are viewed as separate parameters in our system). For instance, color can be considered a three-dimensional parameter because it can be composed of three one-dimensional parameters: either red, green, and blue values, or hue, brightness, and saturation values.

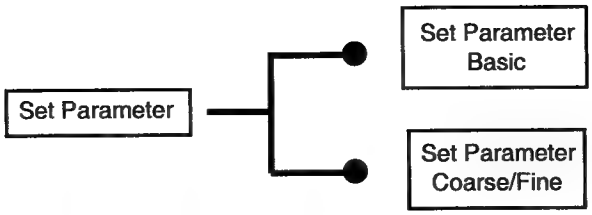

Figure 13. The top level of the parameter setting task model. The task can be specialized as either basic parameter setting or coarse/fine parameter setting.

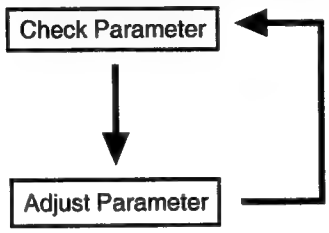

Figure 14. The basic parameter setting task model.

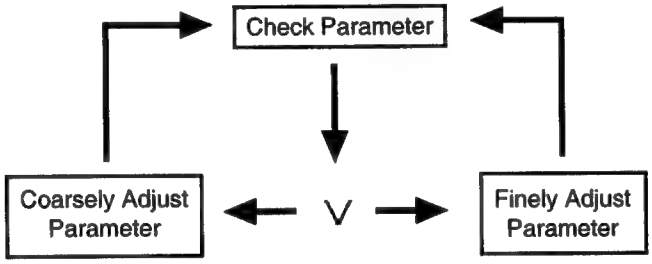

Figure 15. The coarse/fine parameter setting task model.

Value Type

- Discrete
  - Binary
  - OR set
  - XOR set
  - Range
- Continuous

Composition

- One-Dimensional
- Two-Dimensional
- Three-Dimensional

Setting Effect

- Selection
- Performance

Figure 16. Classification criteria for a parameter object.

Finally, the effect of setting a parameter can be one of selection, in which a choice is made that has delayed effects, or one of performance, in which an immediate effect is achieved by the performance or control of a function. An example of this distinction and its influence on an interface is the case of turning on a light; turning on a light by clicking on a checkbox would feel strange and unintuitive because checkboxes are typically used for making choices on a form, not for performing actions.

The second object involved in the task of parameter setting is a display for providing feedback about the value of the parameter. The display can be optional because feedback is often inherent in the control used to set a parameter (e.g., a slider switch indicates a relative setting by its position) or the observable effects of setting a parameter (e.g., volume or brightness). However, sometimes a separate display is needed. For instance, if the real effect of a choice is delayed or expensive to compute, a display can be used to preview the parameter's value before it is actually set. Alternatively, perhaps a parameter will be set remotely and a separate display like a video monitor is needed to see the effects.

The third object involved in the task of parameter setting is a control for setting the parameter. Our criteria for classifying controls is shown in Figure 17.

First, the movement of a control can be smooth, mirroring the continuity of the parameter, or detent, mirroring the discreteness of the parameter. There are two special cases of detent controls: a binary detent control like a light switch, and other detent controls like an old-fashioned TV channel selector.

Next, a control can be persistent or non-persistent. If a control is persistent, it holds the parameter at the value it was last given. If a control is non-persistent, as soon as the user "releases" it, the control returns the parameter to some default value. An example of a non-persistent control is the pitch slider wheel on many synthesizers.

Finally, as mentioned above, a control might or might not provide feedback about the value of the parameter or its effect. If a control does provide feedback, the feedback can be quantitative (e.g., by reading a labeled knob, a user can see at exactly what value the knob is set) or qualitative (e.g., by looking at the relative position of a slider, a user can see if its associated parameter's value is high, low, near the middle, etc.).

17.2 Using the Parameter Setting Interface Library

A designer finds interface objects in our library by answering questions about alternatives provided by the task model described above. The system uses the designer's answers to traverse the decision tree that organizes the library. A sample interaction follows. Because

the parameter is common to the whole task, the system begins by asking the designer to specify the type of parameter to be manipulated.

Movement Type

- Continuous
- Detent
  - Binary
  - Three positions or greater

Persistence of Change

- Persistent
- Non-Persistent

Feedback

- Quantitative Feedback
- Qualitative Feedback
- No Feedback

Figure 17. Classification criteria for a control object.

- XOR Pulldown Menu - marked
- Popup Menu
- XOR ListBox - marked
- Macintosh Radio Buttons
- Radio Buttons with two positions
- TextField

Figure 18. Possible interface objects to use in a computer interface for making a selection from an XOR set of items with a persistent control that provides feedback about the parameter's value.

- Radio Buttons with two positions
- Lever - detent - labeled
- Keylock - labeled
- Rotary Selector Switch - labeled
- Slider - detent - labeled
- Knob - detent - labeled

Figure 19. Possible interface objects to use in a physical interface for making a selection from an XOR set of items with a persistent control that provides feedback about the parameter's value.

Is the parameter discrete or continuous?

Discrete.

Is the discrete parameter binary, an OR set, an XOR set, or a range?

An XOR set.

Is setting the parameter an act of performance or an act of selection?

Selection.

Because the range of the parameter is an XOR set, the system knows that the parameter is one-dimensional and doesn't ask the designer about its composition.

The designer has now identified the parameter type, so the system asks which specialization of the parameter setting task is needed.

Should the user perform the basic parameter setting task or the coarse/fine parameter setting task?

Basic.

The system now moves to the first subtask in the basic parameter setting task, which is checking the parameter's value. The system basically needs to know where the user will get feedback about the value.

Which interface object should provide feedback about the parameter's value (the display, the control, both, none)?

The control.

Finally, the system moves to the other subtask, adjusting the parameter, with which the control is associated. The control's movement type is determined by the parameter's value type (i.e., a binary discrete parameter should have a binary detent control), and the designer has just indicated that the control should provide feedback, so the system only needs to ask about the control's persistence.

Is the control persistent or non-persistent?

Persistent.

After the designer has answered all the necessary questions, the system provides a list of interface objects that can be used for the specific parameter setting task that the designer has specified. The objects that could be used in a computer interface based on the above answers are listed in Figure 18. If the interface is to be implemented in the physical world, the designer can use the objects shown in Figure 19. Our libraries currently provide textual suggestions; ultimately, they will provide downloadable, working widgets for a variety of platforms.

18. Visual Comparison

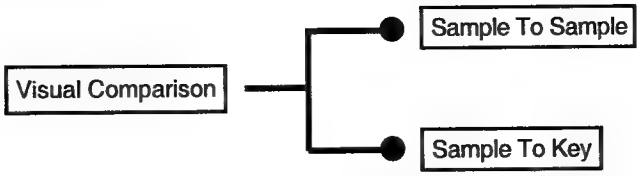

Figure 20. The top level of the visual comparison task model. The task can be specialized as a sample to sample comparison or a sample to key comparison.

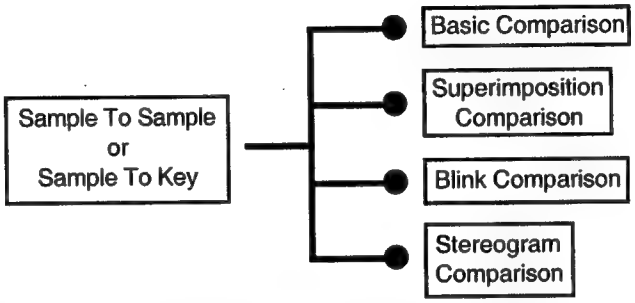

Figure 21. The second level of the visual comparison task. Both the sample to sample and sample to key comparison tasks can be specialized further into one of four types of comparison.

We have implemented a similar model and interface library for the task of visual comparison. Knowledge about visual comparison represented in our task model includes the following:

- Users can either compare two samples or compare a sample to visual key (see Figure 20).
- Visual comparison can follow one of four methods (see Figure 21): basic comparison (i.e., simply looking at two visual objects side by side), superimposition comparison (i.e., aligning a translucent visual object over another visual object to find if the objects match), blink comparison (i.e., aligning at least two visual objects over each other and switching back and forth between them, looking for differences as revealed by apparent motion), and stereogram comparison (i.e., placing two visual objects side by side, defocusing the eyes, and looking for differences as revealed by a 3D effect).
- Visual objects can be either static (e.g., a picture) or dynamic (e.g., video), and either two-dimensional or three-dimensional.
- Visual keys can consist of a single visual object, a linear set of visual objects, a hierarchical set of visual objects, or a continuous range of visual objects (e.g., a color scale).

Figures 22 through 25 show the basic flow between the subtasks for each specialization of visual comparison at a relatively coarse level.

Interface designers can use our library based on the visual comparison task model to find suggestions about interfaces for jobs ranging from matching paint colors to interpreting CAT scans. For example, if the user needs to perform a basic visual comparison between a dynamic, two-dimensional sample and a static, three-dimensional sample, the designer would

find in the library a suggestion for an interface that included at least two viewing rectangles, a VCR-type control for manipulating the dynamic sample, and a rotation control for manipulating the three-dimensional sample. The library would also suggest additional tools for aligning, measuring, zooming, and marking the samples. As another example, Figure 26 shows a possible interface for comparing a static, one-dimensional sample to a discrete linear key.

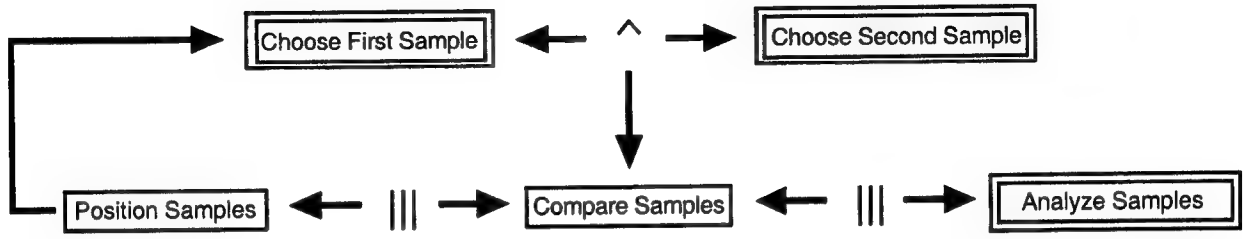

Figure 22. The task model for comparing two samples using either the basic, superimposition, or stereogram comparison methods. First the user chooses the samples to be compared if they have not already been chosen. The user then interleaves the subtasks of positioning the samples, comparing the samples, and optionally analyzing the samples by zooming, measuring, etc. Finally, the user can repeat the process with a new batch of samples.

19. Related Work

Several other projects have also explored semantically-driven approaches to interface design. The previous effort that has taken perhaps the most similar approach to our own work on functionally organized widget libraries, particularly as it relates to the parameter setting task, is the development of Johnson's (no relation) selectors as part of Hewlett-Packard Laboratories' Application Construction Environment, or ACE (Johnson, 1992). Johnson defines selectors as "semantic-based controls."

In the ACE system, interface designers choose widgets based on the type of selection that will be made, and the type of value that will be selected. The system represents several common data types (e.g., numbers, times, Booleans, colors, currency amounts, etc.) and distinguishes between the following selection types:

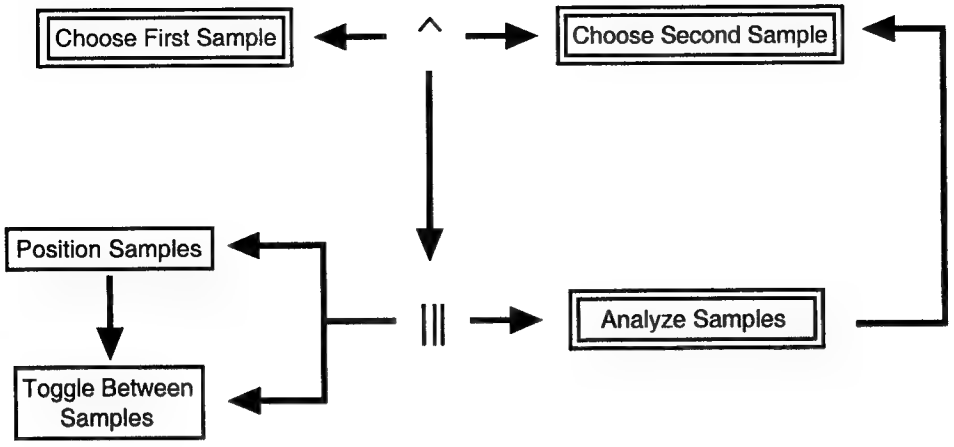

Figure 23. The task model for comparing two samples using the blink comparison method.

Because the samples must be exactly aligned for the comparison to be effective, the order of the subtasks is more tightly constrained than they are in the model shown in Figure 22.

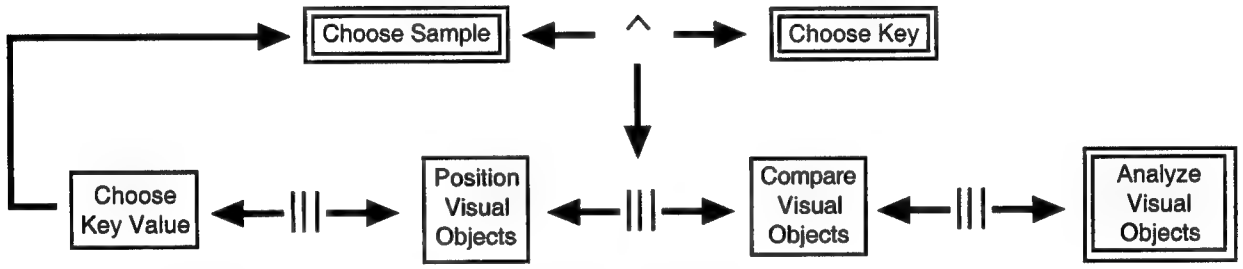

Figure 24. The task model for comparing a sample to a key using either the basic, superimposition, or stereogram comparison methods.

First the user chooses the sample and the key to be compared if they have not already been chosen. The user then interleaves the subtasks of choosing key values, positioning the sample and key, comparing the sample and key, and optionally analyzing the sample and key by zooming, measuring, etc. Finally, the user can repeat the process with a new sample or key.

- Choice of a single value from a discrete set, or 1-from-N choice, which includes the special case of a 1-from-2 choice.
- Choice of multiple values from a discrete set, or N-from-N choice.
- Choice of a single value from a range of values.
- Choice of multiple values from a range of values.

ACE also distinguishes between choosing a data value and choosing a command. Furthermore, each value and choice type has its own group of presenters that determine how the selection mechanism is displayed on the screen. For instance, a color could be displayed as either an actual color or a name, and a 1-from-N choice could be displayed as either a set

of radio buttons or a popup menu. Therefore, in order to put a widget on the screen, designers indicate the value to be set, the way the value is to be displayed, the type of choice to be made, and the way the choice is to be displayed. The ACE system then combines these choices to create a control.

ACE selectors represent a significant advance over most current methods of interface development. Even when using selectors, however, designers are still constructing interfaces widget by widget; and they are still building each interface essentially from scratch. A functionally organized interface library is intended to index interface objects at all levels of complexity, from individual widgets to compositions of numerous widgets. The parameter setting task model we have used to index interface objects in our library is only a single task-model at the lowest levels of a tree of numerous, increasingly complex task models. With a large library of reusable interface objects, ideally many of the choices indicated by the parameter setting task model (e.g., the nature of the parameter's range) would already be constrained by decisions at higher levels.

In addition, the task model that we have used to organize our library provides more extensive indices to parameter setting widgets than does the choice semantics of ACE selectors. Setting a single parameter can require much more than a single control. Consider the Macintosh Color Picker, which contains a two-dimensional color wheel for determining hue and saturation; a slider for determining lightness; three textfields for typing in exact values of hue, saturation, and lightness; and two separate displays for viewing the old color versus the new color. To properly span the space of such complex controls, our model makes explicit such distinctions as classifications for displays, persistence of the controls, etc.

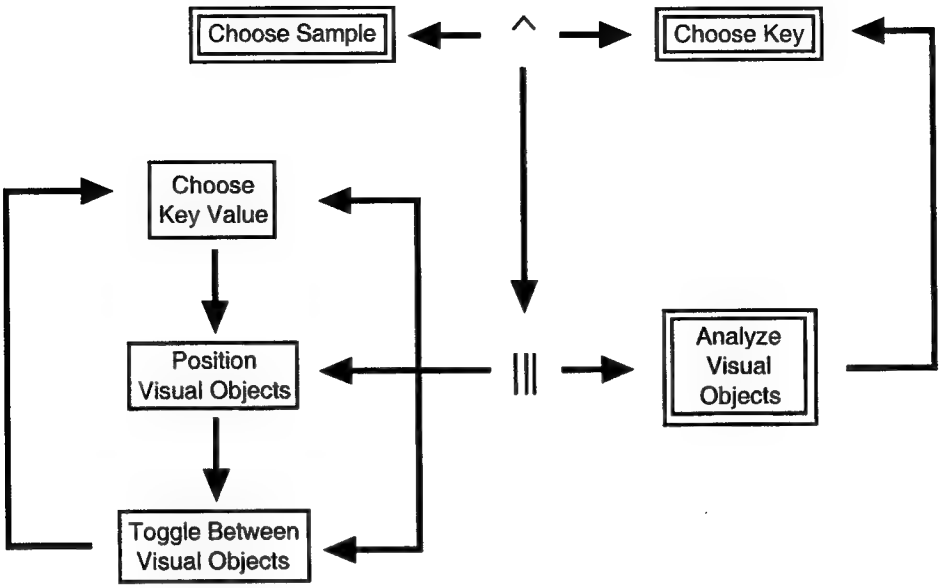

Figure 25. The task model for comparing a sample to a key using the blink comparison method.

Because the sample and key must be exactly aligned for the comparison to be effective, the order of the subtasks is more tightly constrained than they are in the model shown in Figure 24.

In terms of the concept of interface design based on explicit task models, MODEST is related to similar projects such as ADEPT (Johnson, Johnson & Wilson, 1995), Mastermind (Szekely, Luo, and Neches, 1993), and Mecano (Puerta *et al.*, 1994). In more general terms, MODEST, and more specifically the functional organization we propose for interface libraries, grow out of work in case-based reasoning, or CBR (Kolodner, 1993; Riesbeck and Schank, 1989). CBR is an approach to constructing intelligent systems in which knowledge in the form of a specific prior case is stored in terms of the situation in which it is relevant. Using this approach, when a new situation arises, significant features of this situation are used to retrieve the most appropriate prior case. If necessary, the retrieved case is then adapted to the current situation.

In functionally organized interface libraries, interfaces and interface constructs are the prior cases, and the task models represent the situations. Interfaces are stored in terms of the tasks for which they are used. When designers find themselves creating an interface for a new task, they retrieve an interface or construct that is stored under that task and adapt it to their current situation.

20. Conclusions

Our initial goal in constructing functionally organized interface libraries was to facilitate automatic interface generation based on task models in the MODEST system. However, the notion of functional organization is beneficial at any level of technological sophistication. A useful implementation can be as simple as an index in the back of a software tool's documentation that lists available interface objects in terms of their functionality.

In whatever manner a functional organization is used, the key benefit of such an approach is basically this: Interface designers can move intuitively and efficiently from the task they are contemplating to interfaces and interface constructs that are most appropriate for performing that task. Using a functional organization, it is not necessary to hunt through a set of interface objects that have been indexed according to different criteria, perhaps missing an even better interface or making a poor decision along the way.

In addition, better organization techniques for libraries mean larger libraries can be built and utilized effectively. In terms of useful searches, the current organization techniques of alphabetic or object-oriented indexing won't scale up to libraries of any great size. The ability to create and use large repositories of reusable interfaces pushes the field of human-computer interaction closer to the efficiency of engineering methods rather than perpetuating methods based heavily on the skill of the individual designer.

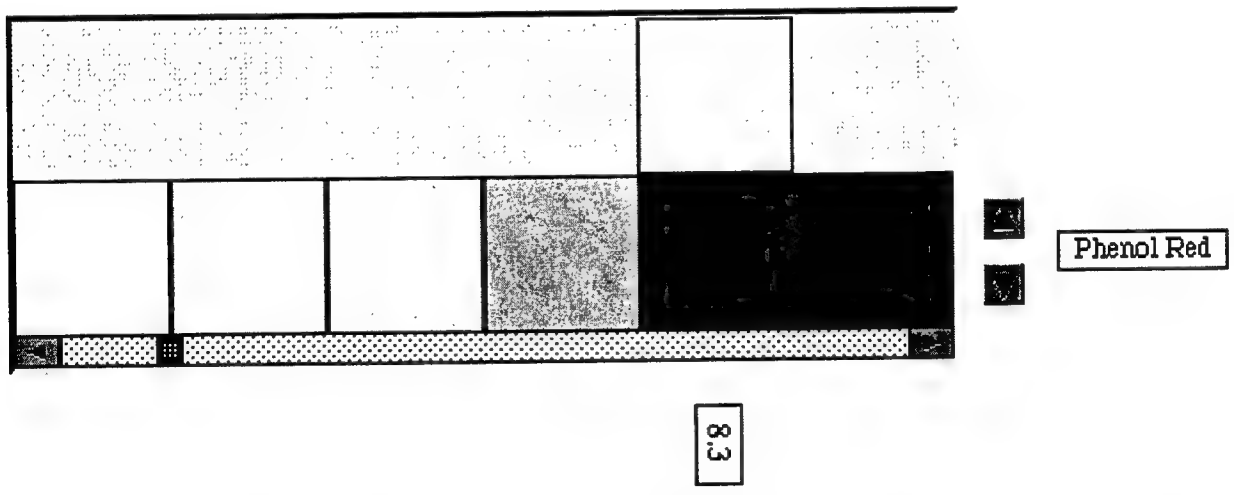

Figure 26. A possible interface for comparing a static, one-dimensional sample (a color) to a discrete linear key (of several colors). This particular interface includes a draggable viewer rectangle for the sample, a scrollable linear key, buttons for choosing a different key, a label to identify the current key, and a label to identify the key value of focus.

Appendix A: State diagrams for selected interface objects

In this section we illustrate the behaviors of selected interface objects and idioms using annotated state diagrams. In these diagrams, state names are in boldface, state transition triggers are in italics and actions on transitions are underlined (when shown). As can be seen, these objects are not pure finite state machines because can they invoke arbitrarily complex functions on state transitions and the triggers may include parameters which are passed on to those functions. The parameters can also be used to filter triggers. We denote this in these figures by showing the parameters in parentheses when relevant, e.g., *transfer(FromContainer)* means accept the *transfer* trigger when the first parameter is the value of the *FromContainer* slot.

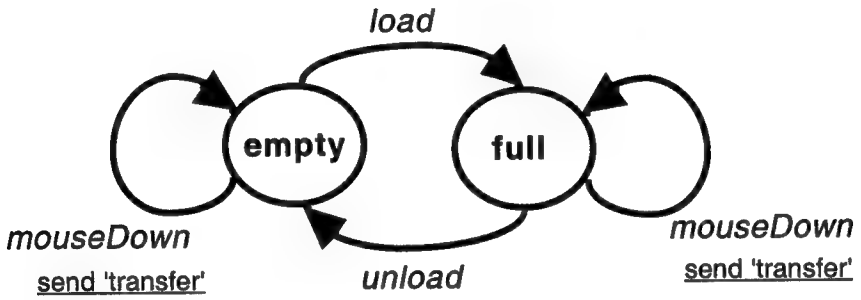

Figure 27. ContainerObject

A container is modeled as a two state object: it is either empty of full of its contents. It responds to the load and unload triggers by executing actions that set or reset its contents and changing state. Because containers are displayed as graphical objects on the screen, they can receive mouse events. Note that instead of responding directly to mouse events, they translate these events into 'transfer' triggers which they pass on to their manager (typically an interface idiom such as ClickTransfer or ClickTransport, q.v.) It is the responsibility of the parent interface manager to send back the semantic triggers 'load' or 'unload' when it is appropriate.

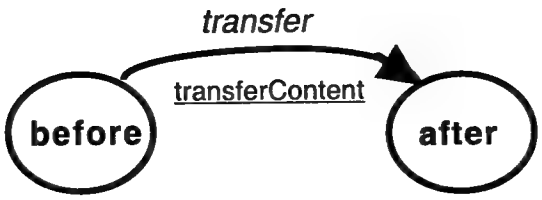

Figure 28. ClickTransfer

ClickTransfer is an interface idiom that transfers the contents of one container to another. It is invoked when the user clicks on one of the containers that is a parameter of the transfer FSM. In response, it sends the 'unload' and 'load' triggers to the containers in its FromContainer and ToContainer roles.

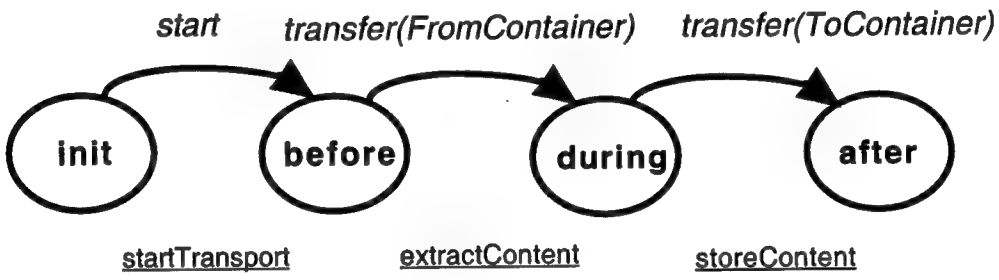

Figure 29. ClickTransport

ClickTransport is a means of transporting a fluid from one location to another, in effect by transferring it from the FromContainer to an intermediate container (represented by the mouse pointer), and finally to the ToContainer. We present this idiom to illustrate the use of parameterized messages or triggers. For example, ClickTransport will only transition from the before state to the during state if it receives a transfer message from the container in its FromContainer role.

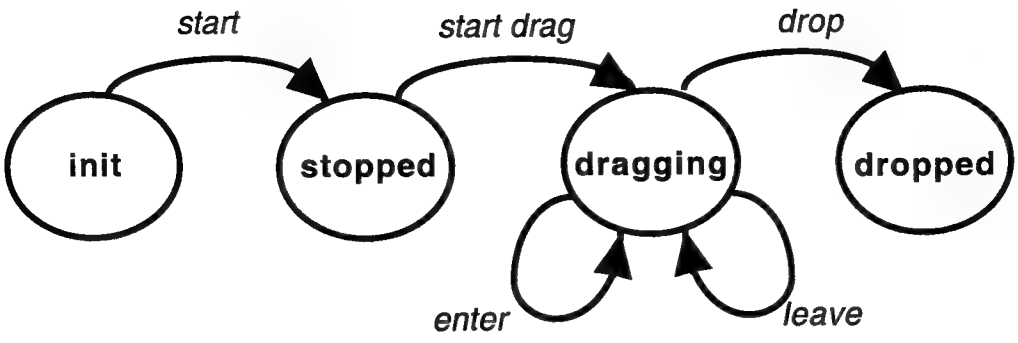

Figure 30. DragOverTarget

Another way to move an object is to drag it and drop it. The drag over target idiom is a slightly elaborated version of this that receives triggers when the dragged object enters or leaves a designated target object. This permits the dragged object to respond to those events as appropriate, as for example when litmus paper is "dipped" in a solution by dragging it over a beaker.

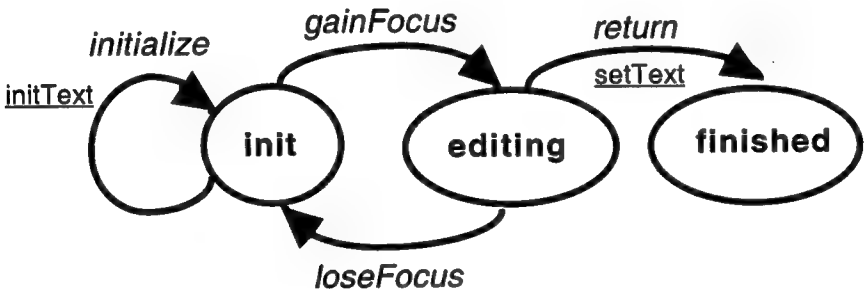

Figure 31. TypeIn

Some user-interface idioms correspond more directly to standard widgets. The TypeIn idiom permits a user to enter and edit text in a text-editing field and returns the text when he presses the carriage return. The initText routine sets up the event receivers to send the gain and lose focus triggers when appropriate.

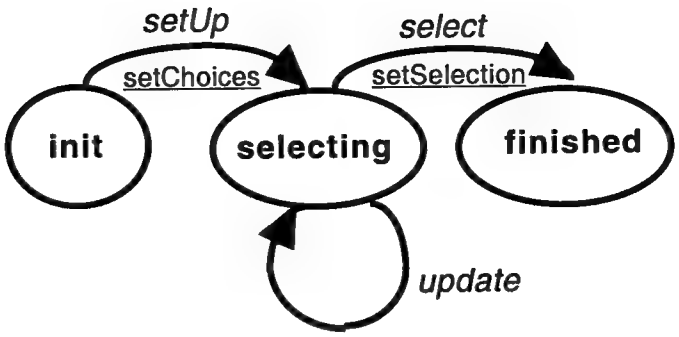

Figure 32. UserSelect

The UserSelect idiom supports the behavior of choosing a value from a list. This works with either a listbox widget or a popup menu.

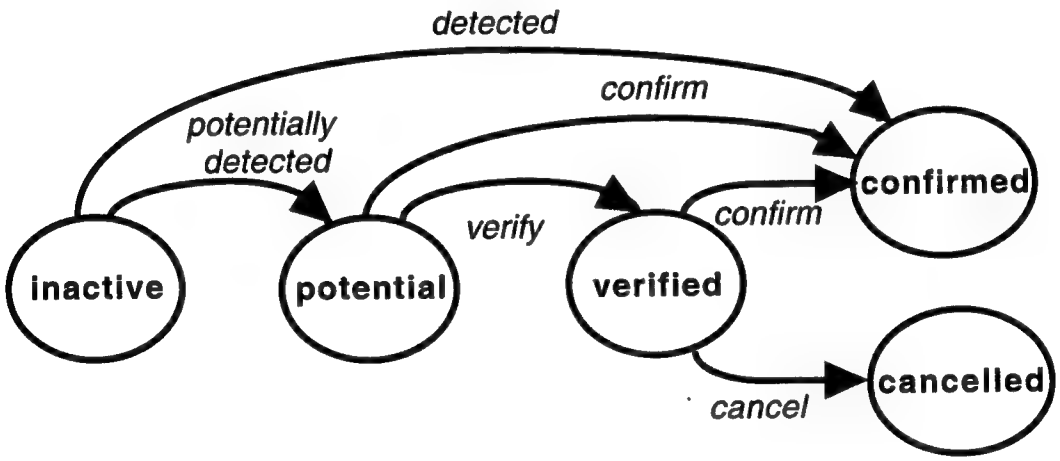

Figure 33. ThreatObj

This diagram shows the states and transitions of a threat object. This behind-the-scenes object coordinates the threat detection task around the states of possible threats on the agenda.

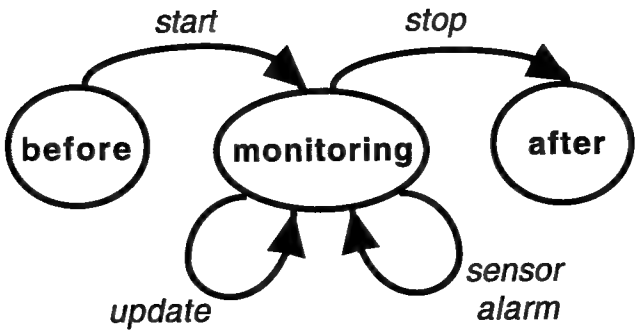

Figure 34. Scanning

Scanning is an interface idiom that implements the monitoring subtask. It is responsible for updating the display based on events such as sensors exceeding their threshold or the user selecting a video channel.

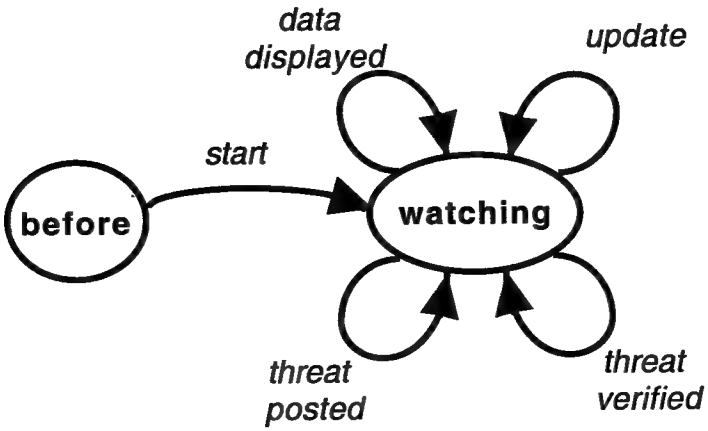

Figure 35. SelectInfoChannel

SelectInfoChannel is an interface idiom that supports viewing information channels on a multiplexed display and determining whether a threat (or any event for that matter) has been displayed since it arose. Note that verification cannot be driven solely by the operator changing the channel, since the relevant channel may already have been selected when the threat arises.

Appendix B: Ontology

In this section, we outline the hiererchy of concepts that are currently represented in the MODEST tool. These concepts are represented as frames with multiple inheritance. In the diagram below, names in parentheses denote additional parents in the frame hierarchy, while names in square brackets denote ScriptX class mixins that augment the frame implementation for certain types of objects.

- WorldObject
  - AttributeType
    - ContinuousAttribute
    - DiscreteAttribute
  - CompositionType
    - ContinuousStuff
      - Water
    - DiscreteThing [FSMActor]
      - ContainerObj
        - Beaker
        - TestTube
      - Machine
      - ControlObj
      - IndicatorObj
        - VisualDisplay
        - Gauge
          - SemiCircularGauge
          - DigitalReadout
          - SemiCircularGauge
        - ChemicalIndicator
        - MultiplexedIndicator
          - MultiInputVideoMonitor [FSMvideoMonitor]
      - Tool [FSMCursor]
        - LiquidExtractionTool (ContainerObj)
          - Dropper
          - Syringe
      - InterpretationKey
        - ContinuousKey
        - DiscreteKey
      - MoviePresenter
      - InterfaceControl [FSMResettableActor]
        - ControlButton
        - SelectionObj
          - Palette [FSMPalette]
            - ButtonPalette
      - Device
        - Alarm
          - VisualAlarm
          - audioAlarm [FSMaudioPlayer]
        - Sensor
          - AutomaticSensor
            - BinarySensor
              - doorSensor
            - LinearSensor
            - ThresholdSensor
              - smokeDetector

2DSensor
videoCamera
AbstractMethod
TaskCombiner
Sequential [SequentialCombiner]
Parallel [ParallelCombiner]
Disjunctive [DisjunctiveCombiner]
Loop [LoopCombiner]
Concurrent [ConcurrentCombiner]
SituationManager
Conditional [ConditionalCombiner]
TaskMethod
OmitTask
PressBttnMethod
WatchMovieMethod
DirectManipulation
Transport
InitiateProcess
MakeContact
PresentInfo
SelectInfo
EnterInfo
Gesture [SceneManager]
DragTransport (Transport)
ClickTransport (Transport)
ClickTransfer (Transport)
ClickTurnOnOff (InitiateProcess)
ClickTurnOn (InitiateProcess)
ClickButton
ClickButtonDoTask
DragOverTarget (MakeContact)
ClickCompareSelect (selectInfo)
UserSelect (selectInfo)
TypeIn (EnterInfo)
WatchMovie (WatchMovieMethod)
Scanning
SelectionInformationChannel
ThreatObj

Appendix C: Sickle Cell Demo Screens

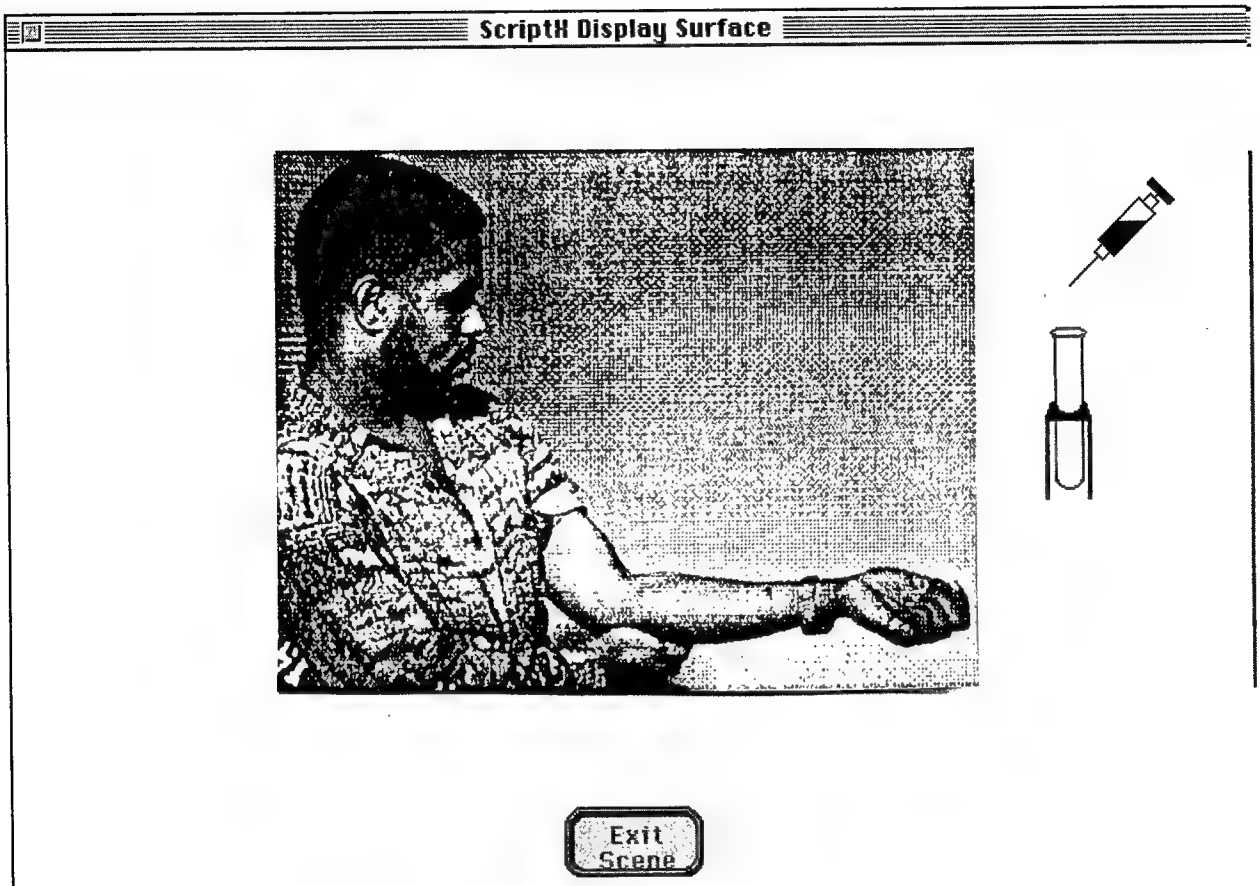

Figure 36. The Sample Scene.

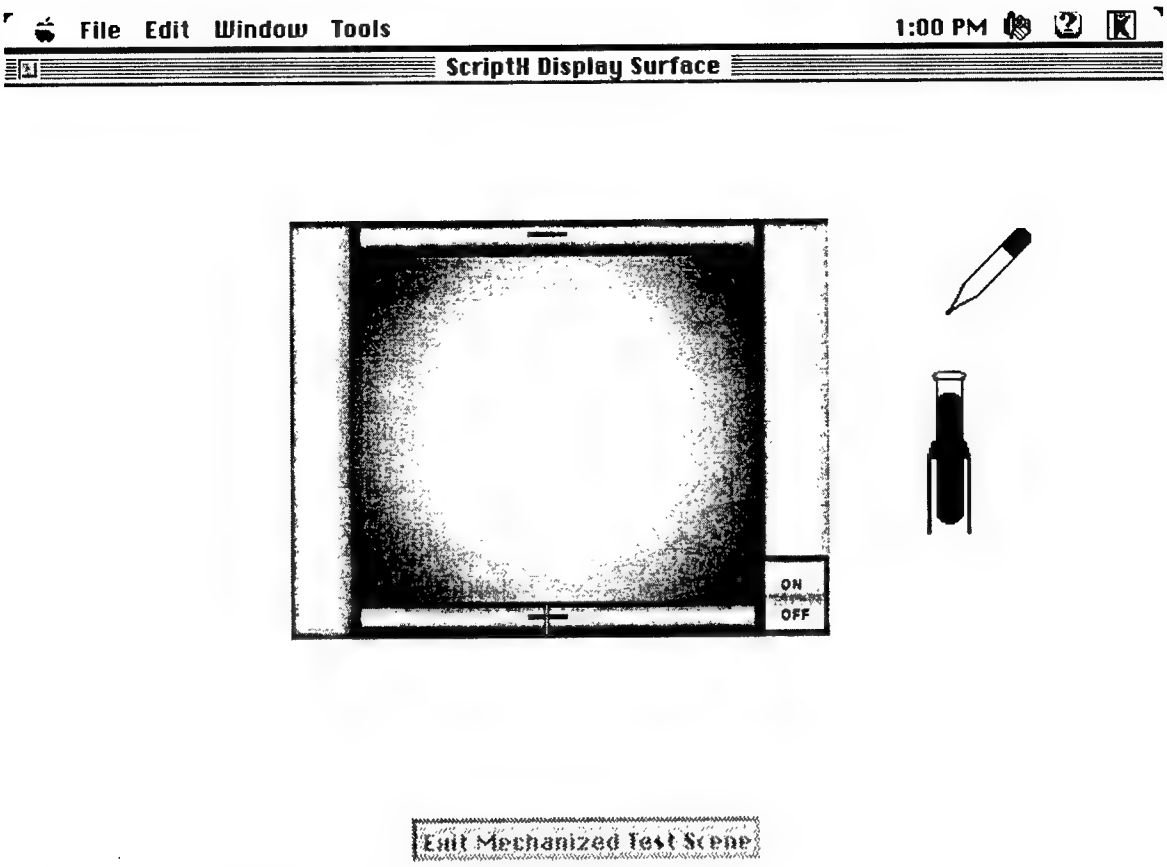

Figure 37: The Electrophoresis Machine (before loading)

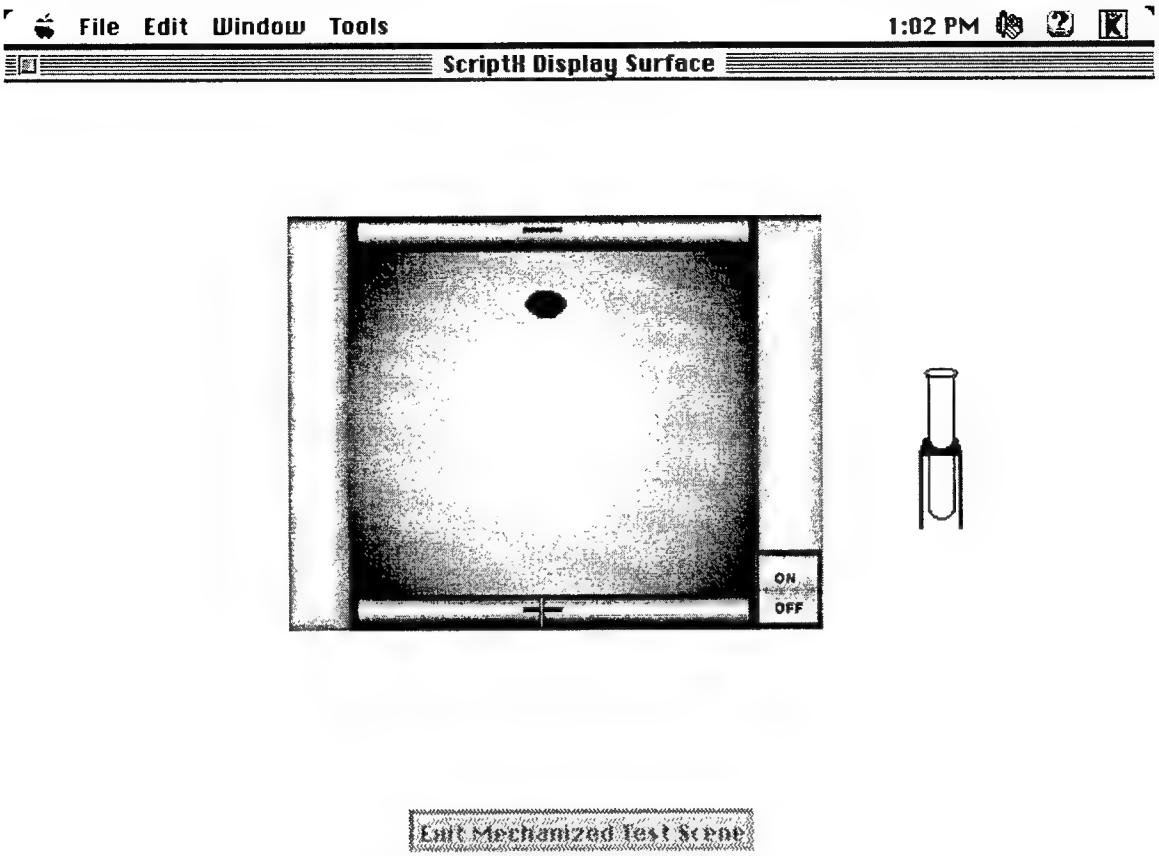

Figure 38: The Electrophoresis Machine (after loading)

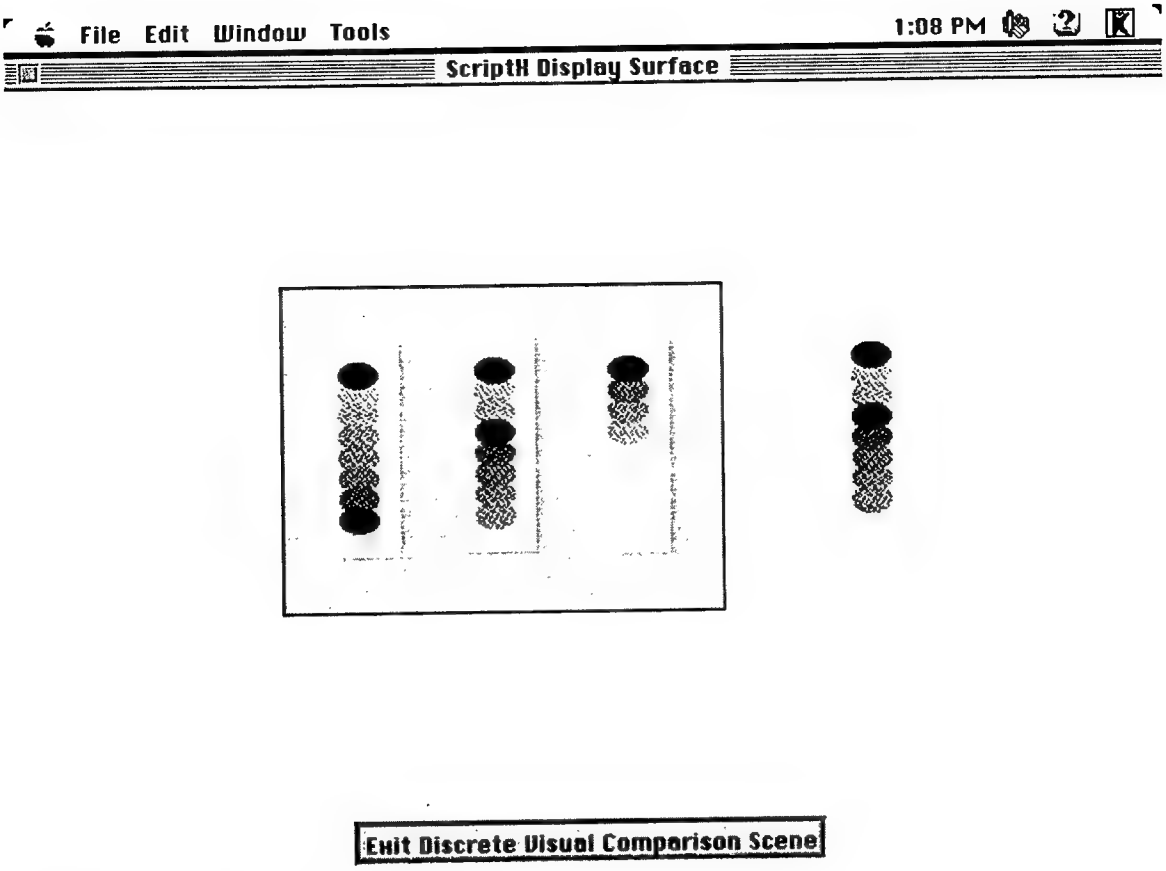

Figure 39: The Interpret Scene

References

Apple Computer. 1995. Macintosh Human Interface Guidelines. Reading, MA: Addison-Wesley.

Bell, B., Bareiss, R., and Beckwith, R. 1993/1994. Sickle Cell Counselor: A prototype Goal-Based Scenario for instruction in a museum environment. *Journal of the Learning Sciences* 3(4):347-386.

Birnbaum, L., and Collins, G. 1993. Towards a general theory of planning and design. Technical Report 44, The Institute for the Learning Sciences, Northwestern University.

Chandrasekaran, B. 1983. Towards a Taxonomy of Problem-Solving Types. *AI Magazine*, 4:9-17.

Cwikla, J.L. *The X Widget FAQ.* <http://www.wri.com/~cwikla/widget/>.

Green, P.E.1986. Issues in the Application of Artificial Intelligence Techniques to Security Systems. In *1986 Carnahan Conference on Security Technology,* 127-132. Lexington, Ky.

Hammond, K. 1989. Case-Based Planning: Viewing planning as a memory task. New York: Academic Press.

Hinrichs,T., Bareiss, R., Birnbaum, L., and Collins, G. 1996. An interface design tool based on explicit task models. In *CHI 96 conference companion,* 269-270. NY:ACM.

Jona, M., Bell B., and Birnbaum, L. 1991 Button Theory: A Taxonomic Framework for Student-Teacher Interaction in Computer-Based Learning Environments. Technical Report 12, The Institute for the Learning Sciences, Northwestern University.

Johnson, J. 1992. Selectors: Going Beyond User-Interface Widgets. In *Proceedings of CHI '92,* 273-279. Monterey, CA: ACM.

Johnson, P., Johnson, H. and Wilson, S. 1995. Rapid Prototyping of User Interfaces Driven by Task Models. In Scenario-Based Design: Envisioning Work and Technology in System Development, J. M. Carroll, ed. New York: John Wiley & Sons, 209-246.

Kaleida Labs. 1995. ScriptX Components Guide. Mountain View, CA: Kaleida.

Kolodner, J.L. 1993. Case-based Reasoning. San Mateo, CA:Morgan Kaufmann.

Puerta, A., Eriksson, H., Gennari, J., and Musen, M. 1994. Model-based automated generation of user interfaces. In Proceedings of AAAI '94, 471-477. Menlo Park, Calif.: American Association for Artificial Intelligence.

Riesbeck, C. and Schank, R. 1989. Inside Case-based Reasoning. Hillsdale, NJ:Lawrence Erlbaum.

Rosch, E., Mervis, C., Gray, W., Johnson, D. & Boyes-Braem, P. 1976. Basic objects in natural categories. *Cognitive Psychology* 18:382-439.

Sacerdoti, E. 1977. A Structure for Plans and Behavior. New York: American Elsevier.

Schank, R. 1982. Dynamic Memory: A Theory of Reminding and Learning in Computers and People. Cambridge, England: Cambridge University Press.

Schank, R. and Birnbaum, L. 1995. ILS Interface Design: Principles and Design Guidelines. Technical Report #62, The Institute for the Learning Sciences, Northwestern University.

Scott, C., Shannon, B., Font, F. and Hatfield, B. 1995. Visual Basic 4 Unleashed. Indianapolis:Sams Publishing.

Sussman, G. 1975. A Computer Model of Skill Acquisition. New York: American Elsevier.

Szekely, P., Luo, P., and Neches, R. 1993. Beyond Interface Builders: Model-Based Interface Tools. In Interchi '93, 383-390. NY:ACM.

Wielinga, B., Schreiber, A. and Breuker, J. 1992. KADS: A Modeling Approach to Knowledge Engineering. *Knowledge Acquisition,* 4:5-53.

Wilensky, R. 1982. Planning and Understanding. Reading, MA: Addison-Wesley.

Williams, S.J. 1987. Alarm Communication and Display Systems for High Security Department of Energy Facilities. In *1987 Carnahan Conference on Security Technology,* 121-126. Atlanta, GA.

Part II: Model-Based Performance Support

21. Introduction

The function of a mixed-initiative planning (MIP) system should be to serve as an intelligent assistant to a knowledgeable human planner. Such an assistant might be expected to provide a variety of services, ranging from gathering information and providing warnings to generating options and carrying out routinized tasks. While there are number of things that an intelligent assistant might need to understand in order to provide such services effectively, the most important of these is the nature of the task it is helping the human planner to carry out. In other words, effective MIP systems should incorporate explicit models of the planning processes they are intended to support.

Such a model-based approach to MIP offers two key advantages. First, MIP entails a significant degree of interaction between human planners and automated planning systems. As in any collaborative activity, successful interaction in such cases will require a shared context in terms of which the acts and needs of the participants can be understood and accommodated. While there are many possible sources of constraint that might contribute to the development of such a shared context, arguably the most central is an understanding of the task being pursued.

The second advantage of a model-based approach to MIP is that it is consistent with the way that successful performance support systems actually operate and are currently built. The ACPT air campaign plan authoring system developed by ISX Corporation provides a good illustration. This system's usefulness in supporting the air campaign planning task stems directly from the fact that its design is based on a detailed analysis of that task.

We are developing an explicit, computer-manipulable model of the ACP task, as well as related planning tasks, along with methods for employing such models to carry out a number of important MIP functions. These include task management mechanisms for utilizing such models to track the performance of the task, including in particular mechanisms for understanding the human user's current planning activities within the context of the task as a whole. The model-based task management system we are developing provides an infrastructure for mixed initiative planning that can serve as a basis for integrating a variety of technical applications into a MIP system. The specific

application that we are developing for such integration is a case-based Air Campaign Planning Advisor providing in-context help and advice to users of ACPT.

22. Problems with Current Technology

22.1 Performance Support Systems

In the current state of the art, the design of successful performance support systems requires painstaking, one-of-a-kind empirical investigations of the tasks being supported. The systems themselves must then be carefully hand-crafted to reflect the results of these task analyses, and they do not, in general, employ explicit models of the tasks they are intended to support. Consequently, the flexibility, extensibility, and generality of such systems are extremely limited, as is their ability to provide timely and relevant in-context help and advice.

Despite these difficulties, the design and implementation of performance support systems based on empirical task analysis constitutes a successful engineering practice. Our work aims to support this practice by providing explicit languages for task modeling, and, eventually, libraries of standardized, broadly applicable models that can be extended by specialization, composition, and other editing operations. The primary goal of such an effort is a significant increase in the functionality of the resulting systems. Explicit task models would permit better tracking of the current state of the task, thus increasing the system's ability to make intelligent choices about what it should be doing, provide in-context help and advice, and so on. At the same time, the use of such models would significantly increase the flexibility of MIP systems by permitting the decoupling of the overall task structure from the specific methods used to carry out different aspects of the task. In other words, explicit task models could serve as an interface between different components of a large-scale performance support system.

There are at least two additional benefits of structuring performance support systems around specific task models as described above. First, such an approach will make it possible to carry out the design and implementation of performance support systems in terms of high level components, thus significantly reducing the difficulty of constructing such systems. Second, the use of explicit models will make it significantly easier to use the resulting performance support systems as a basis for developing effective and realistic training systems, through the addition of scenario-specific data and advice.

22.2 Mixed Initiative Systems

Mixed initiative systems have been developed in a number of application areas, primarily educational and informational. Many of these systems rely upon some type of explicit context modeling to support interaction with the user or student. Perhaps the most widespread approach to supplying the necessary context is user (or student) modeling. However, despite many years of research, it has proven notoriously difficult to construct explicit models of user knowledge and intentions—largely because of the natural idiosyncrasies of different users—and it has proven even harder to apply such models in practice. Thus the approach advocated here is to model the task, not the user.

Another popular approach, dialog modeling, appeals to constraints arising solely from communicative goals, rather than those relating to the successful performance of the task itself. In our view, however, such communicative constraints are secondary, and indeed their successful application in the context of a given task depends upon an understanding of that task. This again argues that we can make the most progress by focusing our attention on the constraints arising from the task itself.

22.3 In-Context Information Systems

We have developed a family of hypermedia systems—ASK systems—which are aimed at providing conversational access to large case libraries for training and performance support. A number of ASK systems have been constructed, in domains ranging from economic history, to the public water supply, to military transportation planning. The largest of these systems, Trans-ASK, contains twenty-one hours of video, in the form of 30-second to two-minute clips, detailing the "war stories" of United States Transportation Command staff personnel concerning their experiences in planning for Operations Desert Shield and Desert Storm, as well as the Somalia and Hurricane Andrew relief operations. These stories provide guidance relating to expert practice and rules of thumb not covered by formal doctrine.

Once a user has located a potentially relevant story in the case library, the organization provided by the ASK model supports efficient and natural access to information that might answer his or her follow-up questions. However, in order to formulate the initial query into the case base, the user must traverse a hierarchical menu system to specify his or her role, task, and problem—a time-consuming and distracting chore for someone in the midst of trying to resolve a problem. The effectiveness of such systems would be significantly enhanced if they were able to

track the user's performance of the task, providing instantaneous access to cases directly relevant to his or her current concerns. The use of a shared, explicit task model both to index information and to track the state of the task would constitute a key step towards the development of such a capability.

22.4 Planning Systems

In order to successfully interact with human users, planning systems must be both *comprehensible* and *advisable*. In other words, the human planner must be able to understand what the planning system is doing and why it is doing it, while the system must at the same time be able to accept advice in terms that are natural to the user. Unfortunately, current AI planning systems can neither represent nor utilize general planning concepts such as "bottleneck," "milestone," "crisis," or even "priority," commonly employed by human planners in reasoning about and discussing planning problems. As a result, these systems can neither explain themselves to, nor take advice from, human planners in the language that those planners find most natural.

People naturally talk about plans and planning processes in terms that refer explicitly to models of such processes. A term such as "milestone," for example, gains its meaning from a rather general model of planning tasks: A milestone is an intermediate, observable objective used to monitor progress towards an ultimate objective. Such terms refer not to any particular planning domain, but rather, to the task of planning itself. In order to interact effectively with human planners, MIP systems must be capable of manipulating such concepts, at least in a limited form. This requires an explicit model of the planning task in which the system is engaged.

23. Task Representations for Model-Based Performance Support

23.1 A General Framework for Integrated Performance Support

Our approach to providing performance support is based on a fine-grained task model that allows the software to track its user and provide in-context help and advice. The first step in designing such a performance support system is to define a task model that identifies the process steps and associated information flows. Given the overall task model, the following items must be defined for each subtask:

- A "tool" that supports or carries out the information processing associated with the subtask. In many instances, this will be a "smart form" that reflects the internal data flow of the subtask.
- Recognition conditions that indicate when the user is engaged in the subtask. While these conditions may in principle be quite complex, in many cases, they will simply involve noting that the user has invoked a tool that is relevant to the performance of that subtask.
- Easily detectable errors. These would include, for example, the invocation of a subtask before its necessary inputs have been generated, the exiting of a subtask before it has been completed, and various sorts of input errors.
- Help and advice. This would include background knowledge relevant to the subtask, step-by-step instructions, justifications, lessons learned, alternatives, examples of subtask performance, common problems and solutions, and so on. This information might be made available to the user in a variety of ways, including, for example, through entry into an ASK system, or by invoking an intelligent information agent.[1]

23.2 Task Representation: Content

The core of our approach is the development of explicit representations for the ACP and similar planning tasks. Ultimately, we believe that there are four critical aspects of a planning task must be explicitly represented:

1. A hierarchical decomposition of the overall task in terms of its subtasks.
2. The methods employed by the system or the human planner to carry out these subtasks.
3. A description of the causal mechanisms through which these methods carry out their intended tasks.
4. Problems that are likely to arise in each subtask, and possible solutions to these problems.

To see how this approach might apply in practice, let us consider a common subtask of many planning processes, the comparison of a relatively small set of different options. Carrying out this subtask involves several steps. First, the states of affairs that will result from carrying out the various options must be projected. Next, these possible states of affairs must be evaluated with respect to their costs and benefits along various dimensions. Then, these evaluations must be compared, and, finally, the results of this comparison must be summarized for use in a subsequent decision-making step (which might result, for example, in accepting one of the options, combining them in some way, or rejecting them all). Thus, the subtask of comparing

[1] Note that the work required to turn a performance support system based on the above framework into an exploratory learn-by-doing environment for training novices is reasonably clear-cut. Doing so requires the addition of:

- A well-defined problem (or set of problems) for the student to solve.
- Additional problem-specific help and advice.

options must itself be explicitly represented in terms of four lower-level subtasks, namely, *project*, *evaluate*, *compare*, and *summarize*.

In addition to representing the subtasks themselves, it is also necessary to represent the methods by which each subtask might be carried out. For example, the subtask of projecting the outcome of pursuing a given option might be accomplished by simulation, by a search for similar situations in an historical database, or by use of an empirically derived probabilistic model. Evaluation of the resulting outcomes might be achieved through the computation of a simple weighted sum, or it might involve more qualitative assessments. Comparison of the resulting evaluations might be numerical (i.e., ">"), or it might be achieved through a more complex operation involving the explicit matching up of comparable good and bad aspects of each situation. Summarization might simply be a rank ordering of options, or it might involve a complex process resulting in an explicit representation of the most salient trade-offs between different options. Each of these methods must be represented in enough detail to distinguish when each would be most appropriate.

Next, the causal mechanisms by which a set of methods is intended to carry out the overall planning task must be represented. At the very least, this must involve the representation of constraints on the kinds of inputs the individual methods expect, and the kinds of outputs they produce, sufficient to rule out sets of methods that are *prima facie* incompatible. For example, among the candidate methods listed above, *probabilistic projection*, *evaluation by weighted sum*, *numerical comparison*, and *rank ordering*, by virtue of their numerical basis, form a natural and compatible set of methods for carrying out the option comparison task. Other, more complex families of methods will manipulate more complex classes of data. In order to help the user determine appropriate and mutually compatible selections of methods, the system must explicitly represent constraints among these methods in terms of standard types of input and output data.

Finally, the common problems that arise in each subtask, and ways of addressing these, must be represented as well. For example, it is not uncommon to discover, in the summarize subtask, that the two top-ranked options compare very closely with each other. To solve the problem raised by such a deadlock, it is necessary to find additional differentiating characteristics upon which to evaluate and compare the two.

- Critiquing and assessment mechanisms, as appropriate.

23.3 Task Representation: Structure and Function

Our previous research in the explicit modeling of planning tasks for learning resulted in complex predicate calculus formulations (see, e.g., Birnbaum, Collins, Freed, and Krulwich, 1990). Such representations seem inappropriate to us in the context of the current project, both because they are difficult for users and designers to understand, and because they are likely to prove cumbersome as a basis for dynamic operations such as task tracking.

Our current approach to task representations is based on a significantly simpler structure. Our initial efforts have simply involved representing tasks as finite state machines, capturing such global constraints as are necessary in terms of annotations on the states. Despite its obvious limitations, such an approach has a number of important things going for it. First, it lends itself to simple graphical representations of tasks in terms of flow diagrams, which are easily comprehended both by users and designers. Second, it supports a simple approach to task tracking, in which transitions between subtasks are signaled by specific interface operations, as described above. Finally, it is compatible with the mechanisms underlying a number of user interface management systems. For these reasons, although we fully expect that it will ultimately be necessary to extend and modify this approach, we believe that it is an extremely useful first step.

23.4 Task Models and Indexing

A key function of the explicit task representations that we are developing is to serve as an indexing framework for previous cases, information, and advice. In particular, by associating hooks into an ASK network with the subtasks to which they are relevant, it will be possible for the system to offer the user focused, relevant help and advice as it tracks him or her through the performance of the task.

Our ultimate goal is to develop a MIP system that *proactively* offers advice, suggestions, and critiques during the performance of the task, by calling attention to *specific* items of information it deems particularly relevant to the user's immediate concerns, *whether or not the user is aware of his or her need for that information.* In order to be proactive in this way, the system must be able to plausibly infer when the user is in need of such a particular item of help or advice. This will require the formulation of a second, more detailed level of recognition conditions aimed at detecting specific opportunities for the delivery of specific help or advice. In contrast to the recognition conditions for subtask entry, these conditions are likely to depend heavily upon domain-specific features of the system's plan representations. This makes the task of formulating

conditions of this sort considerably more complex. We believe that it will nevertheless be feasible to write useful recognition conditions in terms of such features precisely because the range of possibly relevant advice will already be considerably narrowed by tracking the user's current sub-task.

24. The Air Campaign Planning Task Model

We have constructed a model of the Air Campaign Planning (ACP) task that is an amalgam of existing practices, current Air Force doctrine, and procedures that have yet to be fully adopted by the Air Force at large. The central feature of this model, alternatively referred to as *Effects-based* or *Objectives-based* planning, is an emphasis on describing a plan in terms of the effects to be achieved, rather than solely in terms of activities to be performed. Effects-based air campaign planning was first put into practice in the Gulf War in 1991.

Another aspect of the planning model is referred to as *Strategy-to-Tasks*, which denotes a hierarchical decomposition of objectives from the highest level National Security objectives down to the most specific targets and actions in a conflict. This hierarchical decomposition is intended to ensure accountability by providing an audit trail justifying each and every target to be bombed. Making this audit trail explicit is one of the chief design features of the Air Campaign Planning Tool (ACPT). The strategy-to-tasks concept is still evolving and is becoming progressively more accepted.

The scope of our ACP model is limited by its purpose. Because the model is intended to guide and support planners as they work with ISX Corporation's Air Campaign Planning Tool (ACPT), we have chosen to focus on modeling the high-level planning processes, ending with the production of the Master Air Attack Plan (MAAP). Lower-level processes, such as the generation of the Air Tasking Order (ATO), are grouped together in the *Execute* task, and exogenous processes, such as team formation, are not modeled at all.

Figure 40 through Figure 43 depict our current model of Air Campaign Planning. They show a task hierarchy at two levels of detail: Subtasks 3, 4, and 5 are broken out and illustrated in more detail in Figure 41 through Figure 43. Each of these subtasks corresponds to a level of objective decomposition, corresponding to strategic, operational, and tactical activities. The next section describes these subtasks individually. The problems associated with each subtask, for which advice currently exists in the Air Campaign Planning Advisor, can be found in the Appendix.

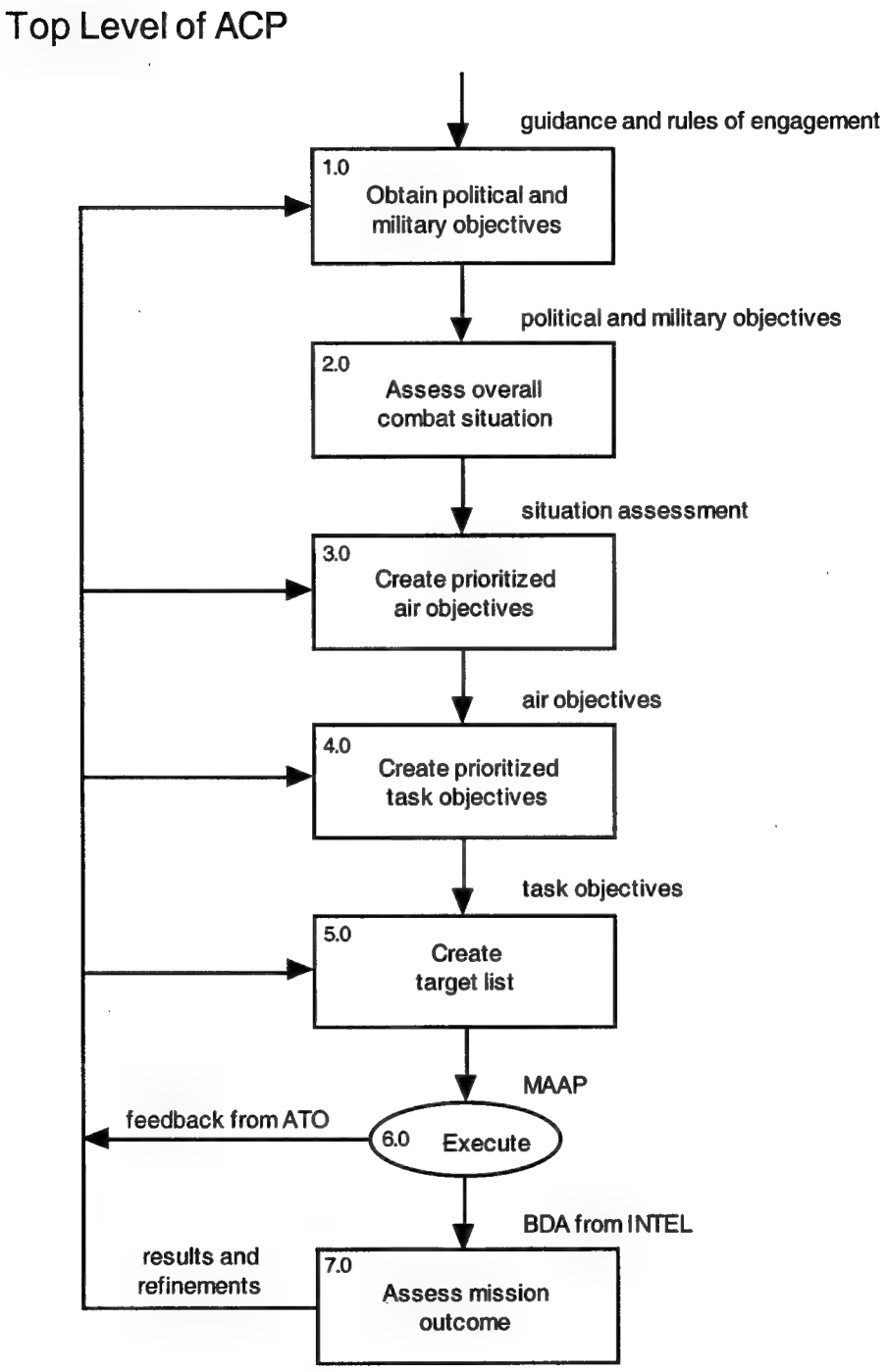

Figure 40: Top Level of Air Campaign Planning

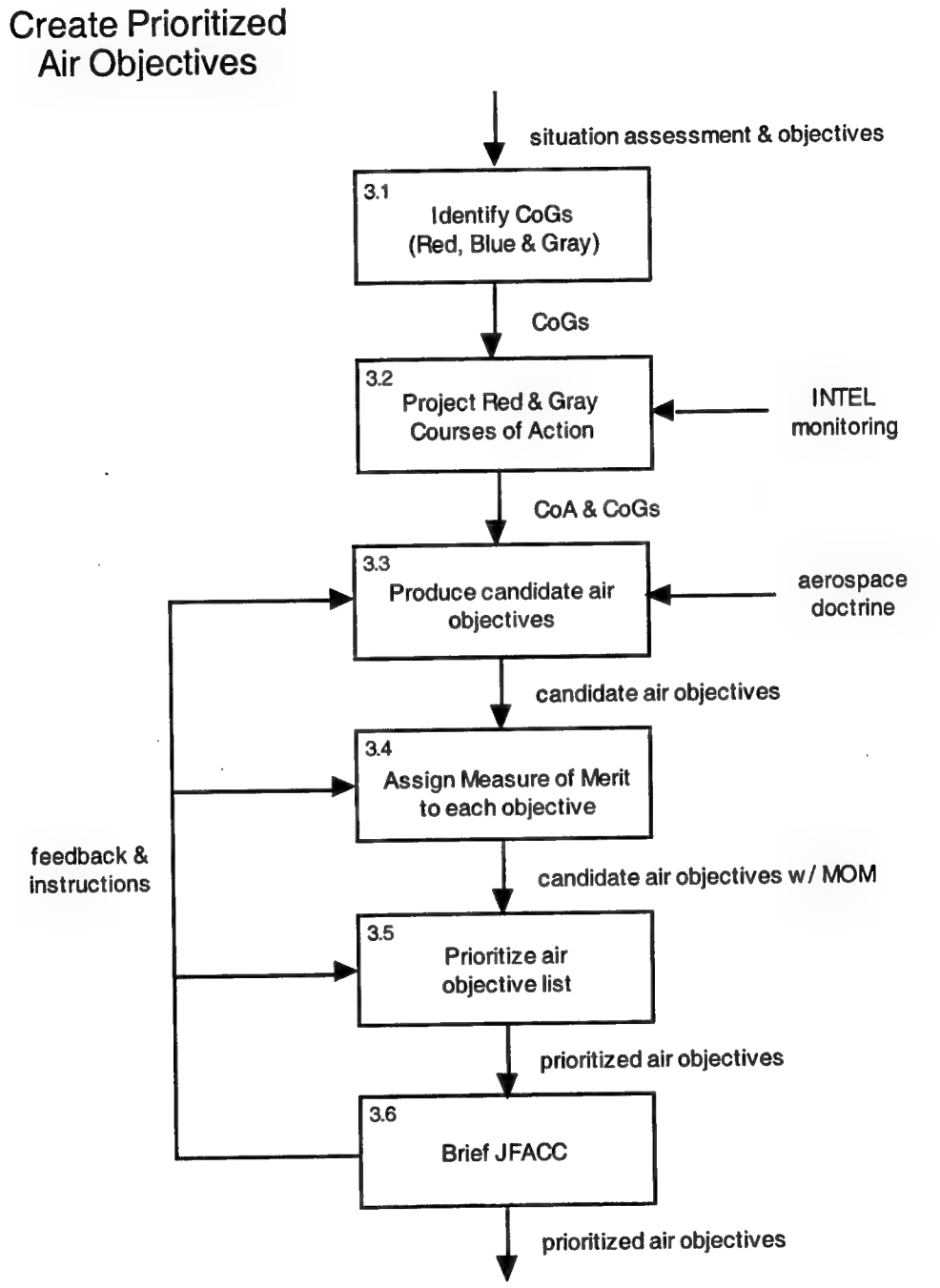

Figure 41: Create Prioritized Air Objectives

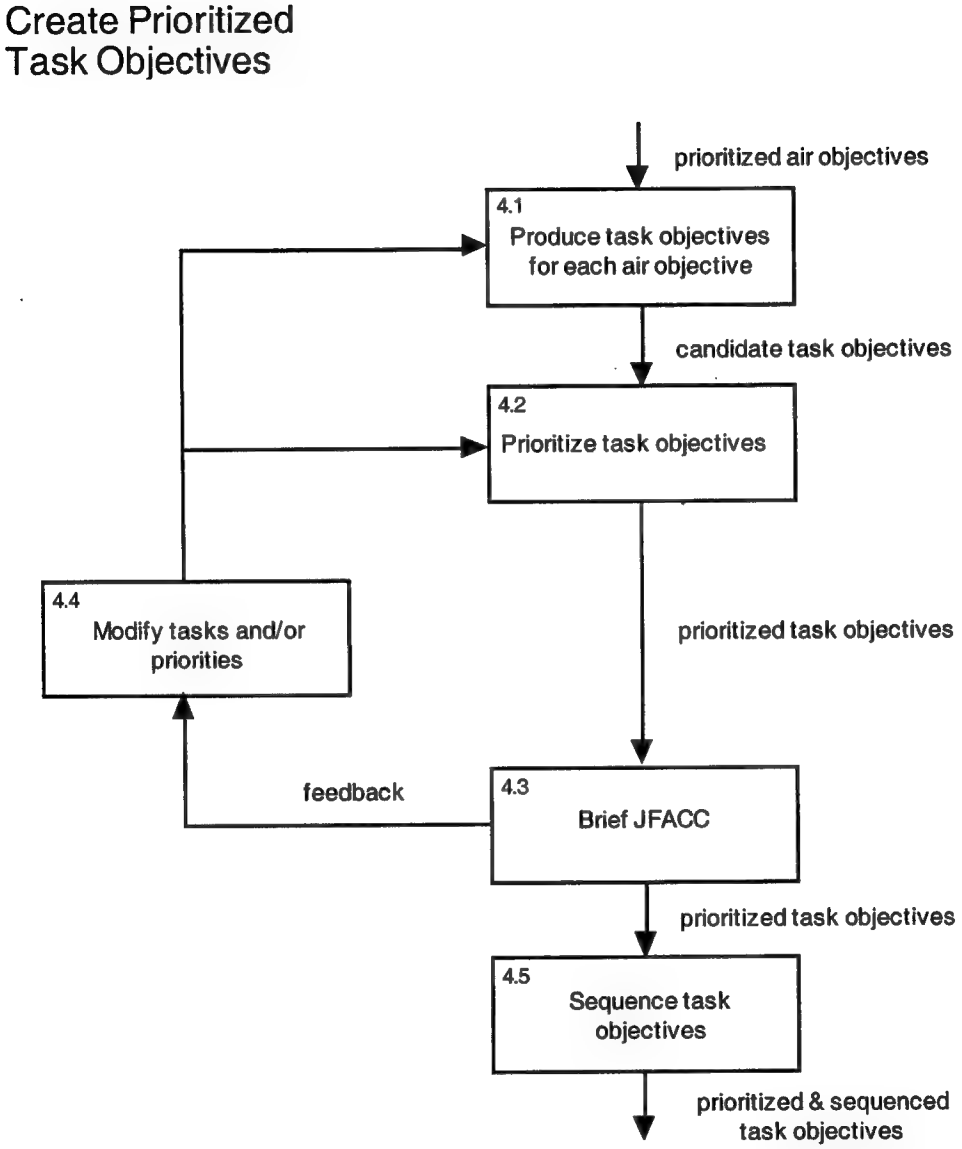

Figure 42: Create Prioritized Task Objectives

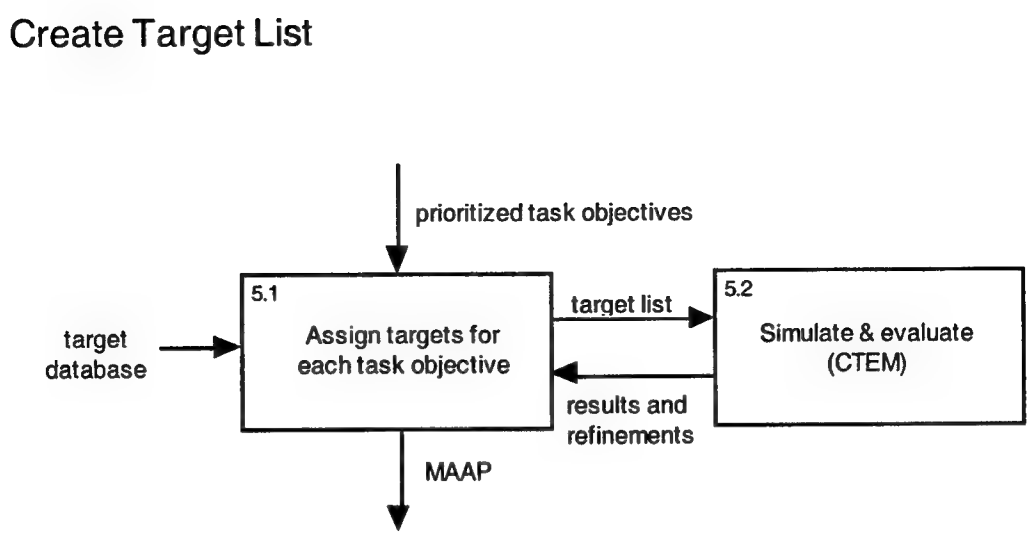

Figure 43: Create Target List

24.1 ACP task model description

Task 1.0: Obtain political and military objectives

The political, or national security, objectives are revised annually. Typical examples of such objectives might be "Preserve world and regional stability," "Defend against threats to U.S. citizens and territory," etc. Military objectives are tailored to a specific regional conflict, for example, "Eliminate Iraq's nuclear production capability." These are typically formulated by the Commander In Chief (CINC) in the theater.

The rules of engagement are derived from international law and from any additional constraints imposed by the President. Commander's Guidance comes from the Joint Forces Air Component Commander (JFACC) or the Commander of the Joint Task Force (CJTF) in the form of a briefing. The task of obtaining guidance and military objectives typically entails a significant amount of constructive interpretation.

Task 2.0: Assess overall combat situation

This is a relatively unstructured task that involves accessing and assimilating INTEL documents on the enemy forces, and assessing the Allied logistics and manpower resources.

Task 3.0: Create prioritized air objectives

Decompose military objectives into objectives that can be carried out through air power. An objective consists of a center of gravity, a desired effect, and metric for measuring success.

Task 3.1: Identify COGs (Red, Blue, Gray)

Using the 5-rings model (leadership, key production, infrastructure, population, and fielded military forces), drive down on the Center of Gravity (COG) corresponding to the military objective to find the key COG that will guide the formulation of air objectives.

Task 3.2: Project Red & Gray Courses of Action

Try to determine what options are open to the enemy and develop your strategy to limit his options. The intent is not to predict exactly what the enemy will do, but to prevent the enemy from taking the initiative.

Gray forces are neutral parties. It is important to understand how they are likely to behave during a conflict: Are they likely to take sides? How will they react if refugees start to flood their borders?

Task 3.3: Produce candidate air objectives

Identify the desired effects that will exert pressure on the enemy's Center of Gravity. This involves analyzing the vulnerabilities associated with the Center of Gravity and carefully phrasing the effect in terms of the appropriate level of destruction. For example, it may suffice to temporarily interfere with (jam) a communications facility, while fuel production must be halted for the duration of the conflict, and weapons of mass destruction must be permanently destroyed.

Task 3.4: Assign a Measure of Merit to each objective

Desired effects represent partially satisfiable goals. A Measure of Merit (MOM) is a quantitative metric to measure the degree of goal satisfaction. Ideally, this will determine not only whether the objective is achieved, but also what kind of INTEL information should be requested in order to assess whether or not the desired effects have been achieved. For example, if a measure of merit simply says that a radar installation will be destroyed, then Bomb Damage Assessment will typically be a satellite photograph showing the bombed out installation. However, if the measure of merit says that there will be no radio emissions from the installation, then the INTEL feedback will more likely be electronic surveillance. The latter provides a more functional measure that can be effective when less severe levels of destruction are feasible or desired.

Task 3.5: Prioritize air objective list

Sort the list of air objectives with respect to their importance in achieving the parent military objectives. The most important objectives will typically be pursued first,

though in the ideal situation, all objectives are pursued simultaneously in order to overwhelm the enemy's systems. This is known as *parallel warfare.*

Task 3.6: Brief JFACC
Describe the plan to the Joint Forces Air Component Commander and revise it based on his feedback.

Task 4.0: Create prioritized task objectives
Decompose air objectives into task objectives. Task objectives may correspond to particular classes of targets (target sets, such as all the bridges in some region) or some particular sector to patrol or enemy fielded force.

Task 4.1: Produce task objectives for each air objective
A task objective is more operational than an air objective. For example, if the air objective is something like: "Locate and track long range field artillery in the Changjon Sector," task objectives might be: "Reconnoiter the Changjon sector for artillery positions," and "Elicit electronic intelligence for artillery in Changjon sector."

Task 4.2: Prioritize task objectives
Sort the tasks with respect to their importance in achieving the parent air objectives.

Task 4.3: Brief JFACC
Present the operational level plan to the Joint Forces Air Component Commander.

Task 4.4: Modify tasks and/or priorities based on JFACC feedback
Revise the plan as required.

Task 4.5: Sequence task objectives
The temporal order of tasks may not directly reflect their overall importance to the strategy. Some tasks are preconditions to others, though they are not intrinsically more important. For example, it may be necessary to take out enemy air defenses before it is possible to reach Command and Control facilities.

Task 5.0: Create target list
Targets for each task objective are selected from a master database. The database provides the target ID, name, target class, latitude & longitude.

Task 5.1: Assign targets for each task objective
Identify individual targets for each target set.

Task 5.2: Simulate & Evaluate (CTEM)
Use the Conventional Targeting and Effectiveness Model (CTEM) simulator to assess the feasibility of a plan. This can be done in two ways: Either input the force

packages available to you and CTEM will return the effectiveness of the missions and losses (attrition), or enter the desired level of effectiveness and CTEM will indicate how much force must be applied.

Task 6.0: Execute

The output of ACPT is a Master Air Attack Plan (MAAP), which is basically a prioritized and sequenced list of targets. This can be fed into the Contingency Theater Automated Planning System (CTAPS) to generate the Air Tasking Order (ATO). The ATO adds a significant amount of detail in the form of route planning, refueling, deconfliction, and schedules for non-force application missions such as INTEL and logistics missions.

Task 7.0: Assess Mission Outcome

Assessing mission outcome is probably the most problematic part of the process. Getting timely and relevant feedback from INTEL depends on asking the right questions of the right people. The amount of raw and processed data is overwhelming, but it has historically taken too long to reach planners and has often been insufficiently focused. Part of the solution is believed to be better use of measures of merit to specify INTEL plans to collect more functionally relevant data. Another part of the solution being pursued is structurally reorganizing Intelligence groups so that they work more closely with strategic planning teams. The Intelligence, Surveillance, and Reconnaissance cell (ISARC) in Vicenza, Italy is one example of this.

24.2 Discussion of ACP model

In general terms, Air Campaign Planning can be thought of as an example of planning with partially satisfiable goals, adversarial planning, multi-agent planning, decision making under uncertainty, and interleaved planning and execution. Each of these categories contributes to the model described above.

Because objectives are partially satisfiable, they require a measure of merit to describe the conditions under which they can be declared achieved. Because the plan involves an intelligent adversary, the planning task involves projecting the Red and Gray Courses of Action. Because the planning process involves multiple planners and a chief decision maker (the JFACC), the task includes a briefing and refinement cycle in which the plan and critiques or suggestions are communicated. Uncertainty is ubiquitous in this process: It shows up in projecting courses of action and in using CTEM to assess the feasibility of the plan. Interleaved execution requires mission assessment and replanning to restrike missed targets.

One critical aspect of planning that is conspicuously absent in the model presented above is reasoning about time. At the middle level of planning described in our model, time appears to play a relatively minor role. Tasks are, of course, sequenced, generally by priority, and occasionally based on preconditions. However, at the level of strategy captured in our model, the intent is to achieve as many objectives as possible in parallel. At higher levels of planning, the campaign may be phased in order to distinguish, for example, the air war from the ground war, or deployment from strategic attack. At lower levels of planning, reasoning about time is critical in determining exactly when aircraft have to take off and what routes they must take to rendezvous with tankers and to hit their targets at their designated Time On Target (TOT).

The model we have presented serves its function of providing an organizing structure for planners to quickly access task-relevant help and advice. We chose to present the model as a simple flow chart rather than as a possibly more complex IDEF0 document because we felt the simplicity was appropriate for its audience and purpose. IDEF0 is the DOD standard modeling format that was developed to support requirements specification. In this case, the audience is not software engineers but planners, and the purpose is not specification, but recognition.

25. The Air Campaign Planning Advisor

Based on the model of ACP described above, and using video clips of experts describing real-world experiences in Air Campaign Planning, we have built a prototype Air Campaign Planning Advisor (ACPA). ACPA is an ASK system, a structured hypermedia system oriented around questions and answers of interest to a problem-solver in a given task domain. ASK networks consist of nodes, with textual, video or graphical content, linked to other nodes. To this extent, ASK networks are similar to standard hypermedia networks.

In standard hypermedia networks, such as Web documents and Hypercard stacks, there is one kind of link. Each link goes from a word, phrase, or icon (the anchor) in the content of one node to another content node (the target). There is no semantics or structure to a link. All links are anonymous, and the same kind of link is used to connect, for example, a word to its definition, an object name to a picture of the object, a concept to an example, a person to their biography, and so on. Not only does this lack of semantics mean that the end user is never sure of exactly what a link will lead to, but, equally importantly, authors of hypermedia networks are not sure what links they should put into the system. As the networks grow in size, they become hard to navigate and difficult to author.

In ASK networks, links are labeled and categorized. Every link is labeled with a question. A link *L* with question *Q* from content *A* to content *B* means "*A* raises the question *Q* and *B* answers that question." The links from content *A* are follow-up questions to content *A*. An end-user does not click on highlighted words or icons: They click on questions they want to ask.

In order to make such a network accessible and authorable, however, it is not enough to label links with the follow-up questions they answer. These follow-up questions themselves need to be coherently organized. Analysis of problem-solving conversations, and empirical refinement, have led us to the following eight Conversational-Associational Categories (CACs) for follow-up questions. These eight categories of links can be grouped into four pairs:

- Links to Context versus links to Specifics.
- Links to Causes versus links to Effects.
- Links to Examples versus links to Contrary Alternatives.
- Links to Opportunities versus links to Warnings.

These categories are used to improve both ASK browsing and ASK authoring. They improve browsing because they enable users to narrow down where to look to find an answer to their current question (as opposed to, say, a simple alphabetical listing). They improve authoring because they help authors to (a) determine where information might be missing in an ASK network, and (b) help them to formulate sensible questions with which to label a given link.

A user interacts with an ASK system using a graphical point and click interface to navigate through the ASK network. These browsers consist of two main parts, a few (one to three) entry "zoomer" interfaces, and the main ASK "lotus" interface.

A zoomer is a graphical map of the content of the ASK network, designed to fit the task-driven needs of the users. By clicking on a particular region of the map, the end user jumps to a particular region in the network. In ACPA, the zoomer is built around the model of the ACP task presented earlier. The user locates information relevant to his or her current task by locating that task in the model.

This takes the user to a video giving an overview of that task, with follow-up questions for that video presented in the "lotus" interface. The lotus interface is so called because it arranges the follow-up questions around the central story, like petals on a lotus blossom. There are eight petals, corresponding to the eight conversational categories. Each petal is a stack of zero or more cards, each card containing one

question. Each stack can be easily browsed, and clicking on a question on a card takes the user to a video or text answer to that question. The questions in the lotus are then changed to reflect follow-up questions for this new information. At any point, the user can click a "Go Back" button to return to earlier answers and pursue other follow-up questions.

This interface enables users to quickly engage in a dialogue with the underlying knowledge base. "Why is ...?", "What else could ...?", "What's an example of ...?"—the kinds of questions that people naturally have in problem-solving settings—are always available. Unlike standard hypermedia systems, the connection between two videos or texts is always clearly specified.

The categorization into eight categories benefits both users and authors:

- For end-users, the categories allow a large number of follow-ups to be spread out spatially into well-defined, consistent locations.
- For authors, the categories quickly show gaps in the network, because missing connections lead to an empty stacks in the browser and sparse fan-out/fan-in in the underlying network.
- For authors, the categories suggest obvious kinds of follow-up questions to develop answers for.

26. Implementation

The standalone version of the Air Campaign Planning Advisor runs on a Sparcstation 20 model 71 with 10 gigabytes of digital video on external hard drives. The fact that the system is eventually intended to be tightly integrated with ISX Corporation's ACPT placed a number of requirements on the hardware and software delivery platform. The primary constraint was that ACPA must run on a Unix workstation, specifically running the Sun operating system. This did not affect content development, because ILS's ASK Tool produces as output an Oracle relational database running on a Unix workstation. It did, however, mean that the ASK browser interface had to be completely designed from scratch; no code from previous browsers could be reused.

In some ways, this has turned out to be a blessing in disguise because it allowed us to consider entirely new architectures for serving ASK system content. Whereas in the past we have constructed custom interfaces for each delivery platform, we found the Unix environment made this difficult to do in a timely fashion. Instead, we decided to use the World Wide Web as our front end, and use Netscape to provide the low-level interface software.

This raised the additional possibility of serving the ASK system remotely over the Internet. While this is not a requirement for the Air Campaign Planning Advisor, we expect it to be an increasingly useful capability in future systems. Currently, the only limitation to true Web-based distribution of ASK systems is the prohibitive bandwidth requirements for transmitting video.

Instead of transmitting video over the Web, it is played at the workstation console through a shareware program called Xanim. Xanim plays compressed and flattened Quicktime movies. Because the player comes with source code, it was possible to customize it for our application in order, e.g., to display the name of the speaker in the window titlebar and to position the control panel underneath the player window. In the long run, we expect to replace Xanim with a Quicktime plug-in for Netscape, at which point ACPA will be capable, in principle, of playing video remotely.

The rest of this section will describe the architecture of the ASK server as implemented in ACPA and in its more general form, and summarize some of the interface design decisions that were made.

26.1 Functional Architecture

Before describing the ACPA architecture itself, we will first briefly review the organization of standard Web-based applications. Figure 44 shows the basic Web client-server architecture. A client browser (either local or remote) communicates with a server via the hypertext transfer protocol (HTTP). The server accepts requests for pages through uniform resource locators (URLs) and transmits them back to the client. These Web pages are files encoded in the hypertext markup language (HTML). Because this architecture is limited to retrieving a fixed set of static pages, it is inappropriate for directly encoding the content of an ASK system, which may be combinatorially explosive.

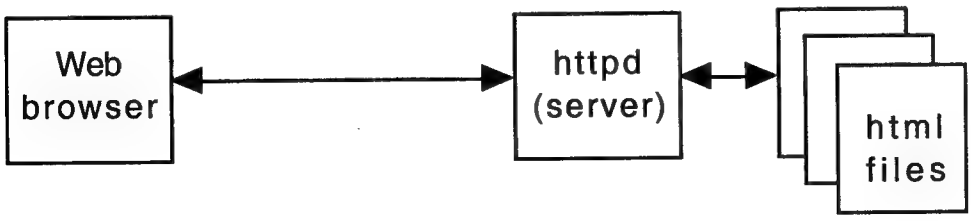

Figure 44: HTML client-server architecture

Figure 45 depicts a Web-server architecture using the Common Gateway Interface (CGI). CGI programs greatly extend the capability of Web applications by allowing URLs to designate programs to be executed, rather than static pages to be retrieved. The CGI program generates an html page on the fly and returns it to the client browser. The most common use of CGI is as a front end for databases and other legacy systems. Such programs translate queries or form input into a database transaction language, such as SQL. The CGI program is usually written in C or Perl and is often quite small.

One of the problems with using CGI to access an ASK system database is that http is fundamentally a *stateless* protocol. This means that every URL is an absolute query; the server doesn't know from one transaction to the next whether it is talking to the same client, much less what page the client is looking at. An ASK system is fundamentally based on navigating through questions and stories relative to the current story. Thus, the first challenge in designing and ASK server is determining how to encode the current state of the browser, as well as its previous history in order to support backtracking to previous stories.

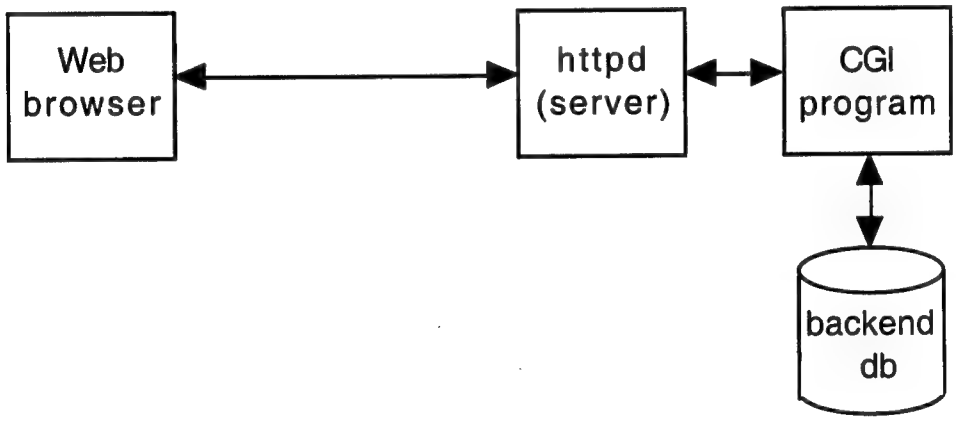

Figure 45: CGI architecture

Our solution to this problem is to encode the state of the browser in the URL itself. A URL that invokes a CGI program can contain variables that serve as arguments to the program. We extended the URL to encode the current story, the question and CAC link being traversed, the history of previous stories visited, and the name of the temporary file that will be returned by the CGI program as its output.

The second challenge in designing an ASK server is efficiency. Whereas most CGI programs are lightweight processes that do simple text manipulations, the ASK system API entails significantly more complex computation. Each page that is generated may involve many database queries, rather than just one. Consequently, the ASK system

API is implemented in Lisp. Unfortunately, Lisp is decidedly a heavyweight process. It takes a relatively long time to launch Lisp, compared to a small C or Perl program. Moreover, while database transactions are reasonably fast, there is significant overhead associated with establishing a database connection.

Our solution is to maintain one Lisp process that is blocked or idle until a URL request is made, at which time it wakes up, processes the request, and blocks again. The database connection remains open for the lifetime of the process, so this cost is amortized. The CGI program that is invoked for each URL request is a small C program, called relay that wakes up the Lisp process, passes on the request, and blocks itself until the result is available. This process necessitates some intermediate synchronization mechanism between the CGI relay program and the Lisp process. The full ASK-server architecture is illustrated in Figure 46.

One additional challenge in designing an ASK server is supporting multiple simultaneous users. Although not a strong requirement in ACPA, future ASK systems will need to support near-simultaneous requests if they are to support distributed access. The difficulty here is that if requests arrive while the Lisp process is generating a page, they could be ignored because the process is a unique resource. This is not a problem for a single-user system because they are unlikely to make requests that quickly, but multiple independent users very likely would.

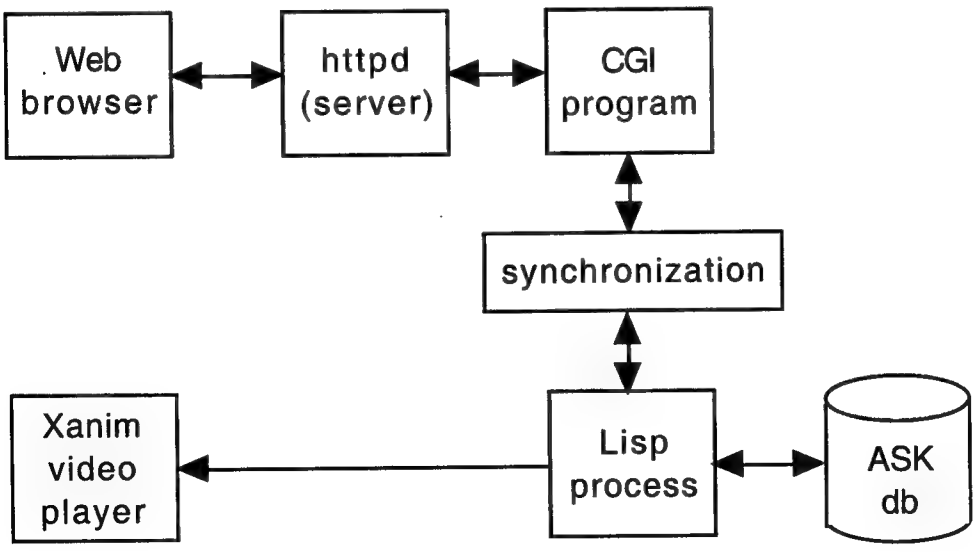

Figure 46: ASK server architecture

Our solution is to use Unix fifo buffering in the synchronization mechanism. The full process works as follows:

1. When a client sends a URL designating a CGI request, it invokes the relay program.
2. The relay program generates a temporary file name to hold the HTML file that will be returned, and adds this file name to the URL.
3. Relay then stores this extended URL in a Unix fifo queue and uses a semaphore to wake up the Lisp process. It then blocks itself by trying to read from the temporary HTML file.
4. Lisp becomes unblocked, reads the extended URL from the fifo, accesses the database, generates the HTML output and writes it to the temporary file designated in the URL. The Lisp program then returns to the top of its main loop and blocks again, waiting for another request.
5. Once the output is written to the temporary file, the relay program unblocks, relays the output back to the httpd server (and on to the client), and erases the temporary file.

Notice the importance of the temporary file name in the URL. If there are multiple requests in the fifo, each request will correspond to a different CGI process and will have a different output. The temporary file guarantees that the right process will be woken up at the right time, and will get the right output.

One other architectural detail that merits discussion is the ASK system database itself. In the development environment, this is an Oracle relational database which actually resides on a remote machine. In the delivery environment, this database is exported to a file format used by Neologic's NeoAccess. The purpose of this is that programs compiled with NeoAccess not only have a smaller footprint than an Oracle server, but also do not require a runtime license fee. The ASK system API was designed to make it relatively easy to change the back-end database in this manner.

26.2 Interface Issues

While the interface for the Air Campaign Planning Advisor looks similar to previous ASK browsers built at ILS, it is basically a Web page and consequently follows the principles for good Web design. Many of these principles are aimed at minimizing the amount of information that must be transmitted over the network, in order to maximize speed.

The first of these is to minimize the size of the images that are transmitted. This is done by tiling the page with small graphics and using a background wallpaper rather than using a single large image. Small graphics tend to be rendered faster, but more

importantly, they can be cached independently, thereby amortizing their transmission cost.

Small graphics rules out the use of imagemaps as a means of dispatching requests. Instead, the ACPA ASK browser is implemented as a table containing independent buttons for each CAC link and for other controls, such as the glossary and the 'go back' button.

In fact, even the task model zoomer is implemented as a table of buttons. The zooming interface is simply a graphical presentation of the task model shown in Figure 40 - Figure 43. Those figures are sliced into horizontal (and sometimes vertical) strips, each of which is functionally a button. A leaf task takes the user to a menu of questions from which he can choose a story.

Question menus are implemented in ACPA as full pages containing a list of questions as links. Ideally, it would be preferable not to entirely obscure the previous page, but this compromise was necessary because modal dialogs are not available in HTML.

26.3 Discussion of the Architecture

The use of the Web as a user interface has proven to be an appropriate design decision. It has not only simplified the programming effort required, but it has also provided a measure of platform independence and has the added benefit of supporting multiple remote clients.

We expect future systems to increasingly use a version of this architecture for the reasons outlined above, and to simplify software maintenance, updates, and even to support closer interaction between the end users of a system and the program developers. For example, in the case of the Air Campaign Planning Advisor, there is a comment button that users can select when they want to provide feedback on the program. The comment button access a URL at a machine at the Institute for the Learning Sciences, which generates and returns a comment form. When the user fills out the form at his machine (which may be anywhere in the world), it is immediately returned and stored in a file at ILS. Such instant feedback may prove to be sufficient justification by itself for designing systems for Web-based delivery.

27. Conclusion: Implications of ACP for Theories of Planning

27.1 Discussion

Interviews with expert air campaign planners for the purposes of task modeling, and to gather video for inclusion in ACPA, have yielded a wealth of empirical detail about a complex, real-world planning practice. From these details, two major themes have gradually become apparent, with broad implications for theories of planning.

The first is that planning takes place within the context of a great deal of ongoing activity, processes, and procedures. Thus the planner is not responsible for generating all of the behavior that the system carries out—perhaps not even most of it. The generative nature of the behavior we observe stems not from the generativity of planners, but from loops in ongoing processes and procedures. The air operations planning process is a good example of this. The MAAPs produced by the Black Hole during the Gulf War served as input to the ATO generation process, and led to an intelligent deployment of air power. But if the Black Hole had been blown up early in the War, *ATOs would still have been generated, and missions would still have been flown.* It's just these processes would have been directed much less intelligently, and to much less useful effect.

The upshot is this: Planners sit on top of, and direct, a large number of ongoing processes with control loops of their own. They work not by specifying particular actions to be taken by the system, but by manipulating the parameters of these ongoing processes to cause those actions to occur. Brooks (1991) has made this point with respect to planning of physical activity. In air operations planning, we see the same sort of thing in a social context, where the ongoing loops are administrative or bureaucratic processes such as ATO generation. Similar observations have been made by Agre and Chapman (1987) and others working on situated activity. Brooks, of course, has argued that this kind of model means that intelligent behavior does not need representation. We believe, on the contrary, that a planner must have a good model of the structure and function of the ongoing processes it is attempting to control.

The second broad theme emerges as a corollary of the first. If planners must work by manipulating the parameters of ongoing bureaucratic systems, then much of their creativity and intelligence must be directed towards making these systems produce

good results in novel environments. In other words, a great deal of planning involves devising intelligent "work-arounds" for handling unforeseen situations.

A striking example of this was provided by Col. Dave Deptula in his manipulation of the ATO generation process during the Gulf War. Col. Deptula and his staff would deliver a MAAP every day to the personnel involved in generating the ATO. After delivering the MAAP, they would continue to refine it based on additional information and, in part, simply because they had additional time to reflect on their task. They found that if they delivered these changes incrementally to the people building the ATO throughout the day, it slowed down the ATO generation process too much.

In response to this problem, they devised the following protocol: They would hand the MAAP to the people building the ATO every day by noon. These personnel, in turn, pledged to deliver the ATO by 6 p.m. if they weren't asked to make any changes. Col. Deptula and his staff continued to work on the MAAP, and, when the ATO was delivered, they added a number of change orders to it, to reflect their additional planning. However, there was a practical limit to the number and scope of the changes they could make, and, in addition, they could make no further changes after 6 p.m., since the air crews flying the missions needed the time to plan and prepare.

In response to this constraint, Col. Deptula hit on the following strategy: He set aside a number of reserve aircraft. Each day, these reserves were put through the ATO process with default targets. In this way, they were deconflicted, assigned entry and exit points into and out of Iraqi airspace, and given squawks and tanker track assignments. If no better targets were determined, these aircraft would attack their defaults. But if Col. Deptula and his staff found particularly important targets on short notice, these reserves were prepared to go to new targets on as little as 3 hours notice.

Deptula's manipulation of the ATO generation process reminded one of us (LB) of the steps he had to go through to buy a car. The dealer first tried to determine whether a suitable car was already on the lot, or at least in the possession of a neighboring dealer. When this turned out not to be the case, he used a computer system to gain access to the production schedule for the model in question. Most of the cars in the schedule were already assigned to individuals or dealers. But a few of them were in reserve. These were completely specified cars; the goal was to find the closest matching car to be built the soonest, and tweak its composition just a little, to match the buyer's requirements. Most parameters (engine, body style, etc.) were already set; only a few (color, interior package, etc.) could be changed.

What these two situations have in common is something like this: A computation to produce some assignment of resources to objectives in a reasonably optimal way, such as production scheduling or generating the ATO, requires that a large number of input constraints be provided in advance in order to be executed. However, there will be cases in which, because of new information arising during or after the computation is executed, some of these constraints change. One solution is to put a small percentage of *reserve* resource into the process; these will be assigned to default objectives, which can then be changed if necessary after the entire computation has been run. Although the assignment may be slightly less optimal than would otherwise be the case, this "work-around" adds the flexibility that is otherwise lacking in the process.

This strategy is an example of a process transformation. We believe that there are scores, perhaps hundreds, of transformations of this sort, aimed at resolving problems and impasses that arise during the planning process, and that a great deal of planning knowledge exists in this form. Much of our work in the future will be aimed at uncovering and codifying these transformations, at least to the point where they can be provided as advice to human planners upon encountering new problems or deficiencies in existing practice.

27.2 Current Status

ACPA has currently been deployed at the Air Force Research Laboratory (Rome Research Site) and the USAF Air Ground Operations School (AGOS) at Hurlburt Air Force Base. At AGOS, the system was used as an adjunct part of the Hunter Warrior exercise at the C2 Battle Lab, which involved an air campaign planning staff performing command and control of air combat and combat support forces during a simulated conflict. After the exercise, the system was left behind for further use and evaluation by AGOS personnel.

Appendix A: ACPA Screens

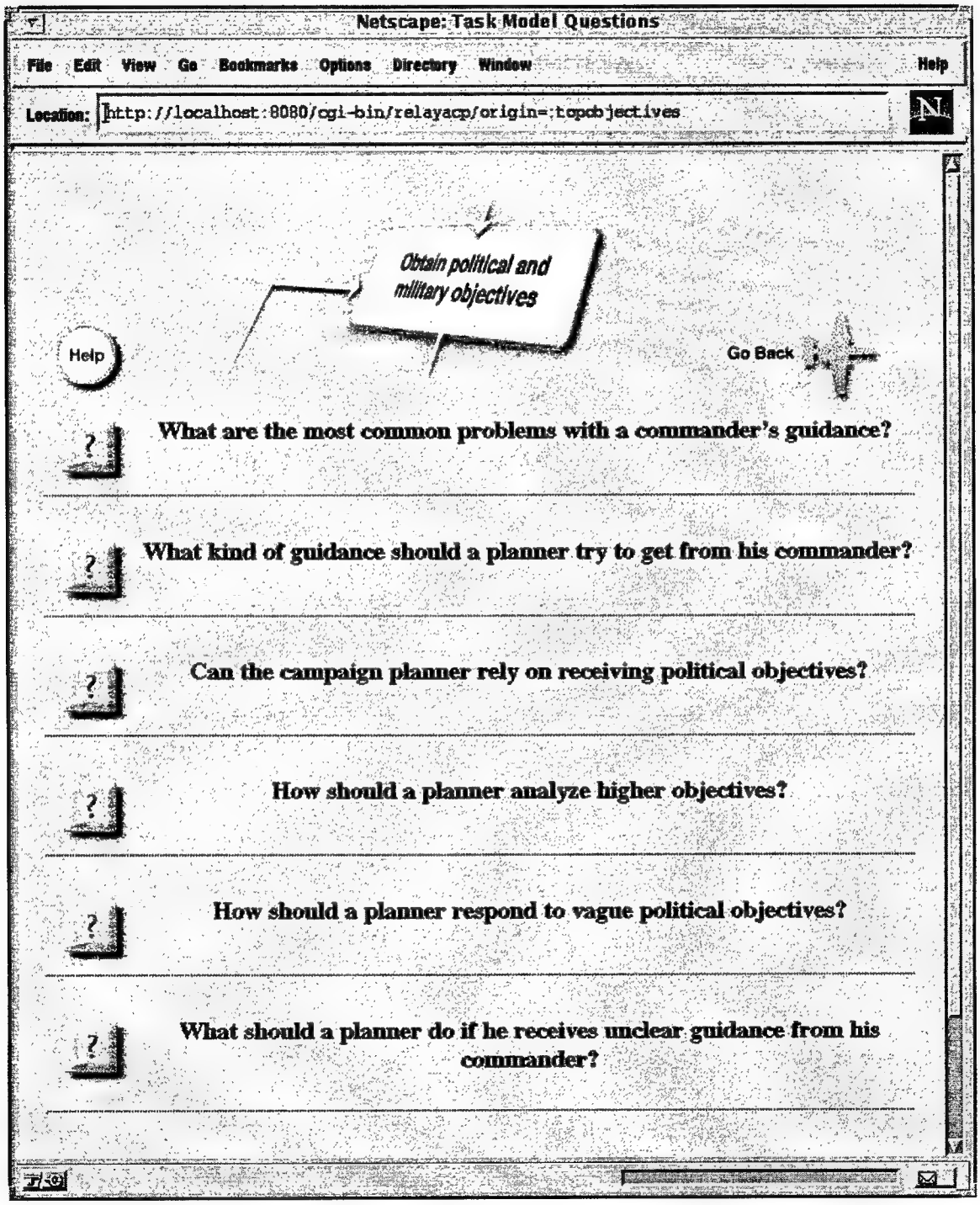

Figure 47: Task Model Questions

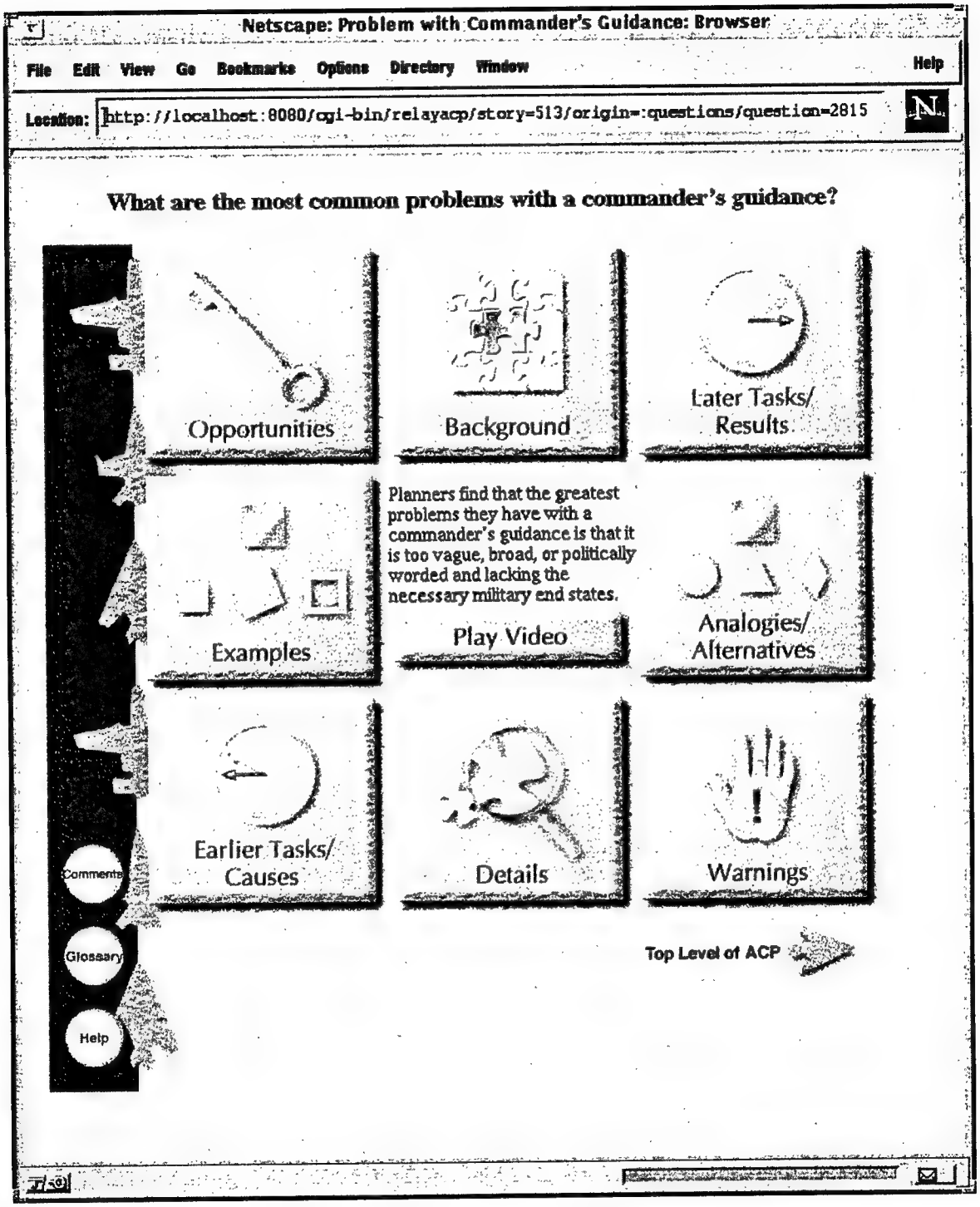

Figure 48: ACPA Ask Browser Interface

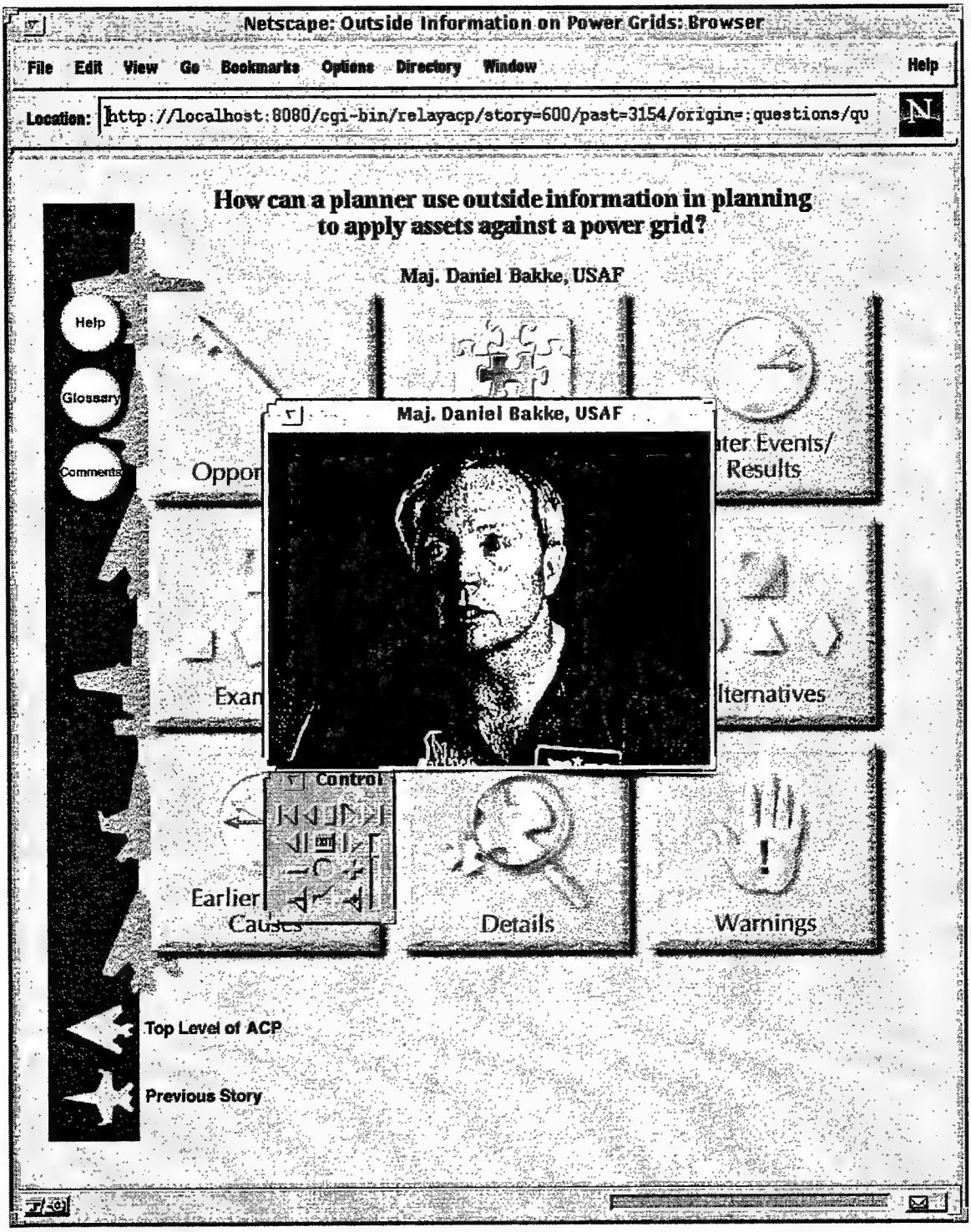

Figure 49: ACPA Browser with Video

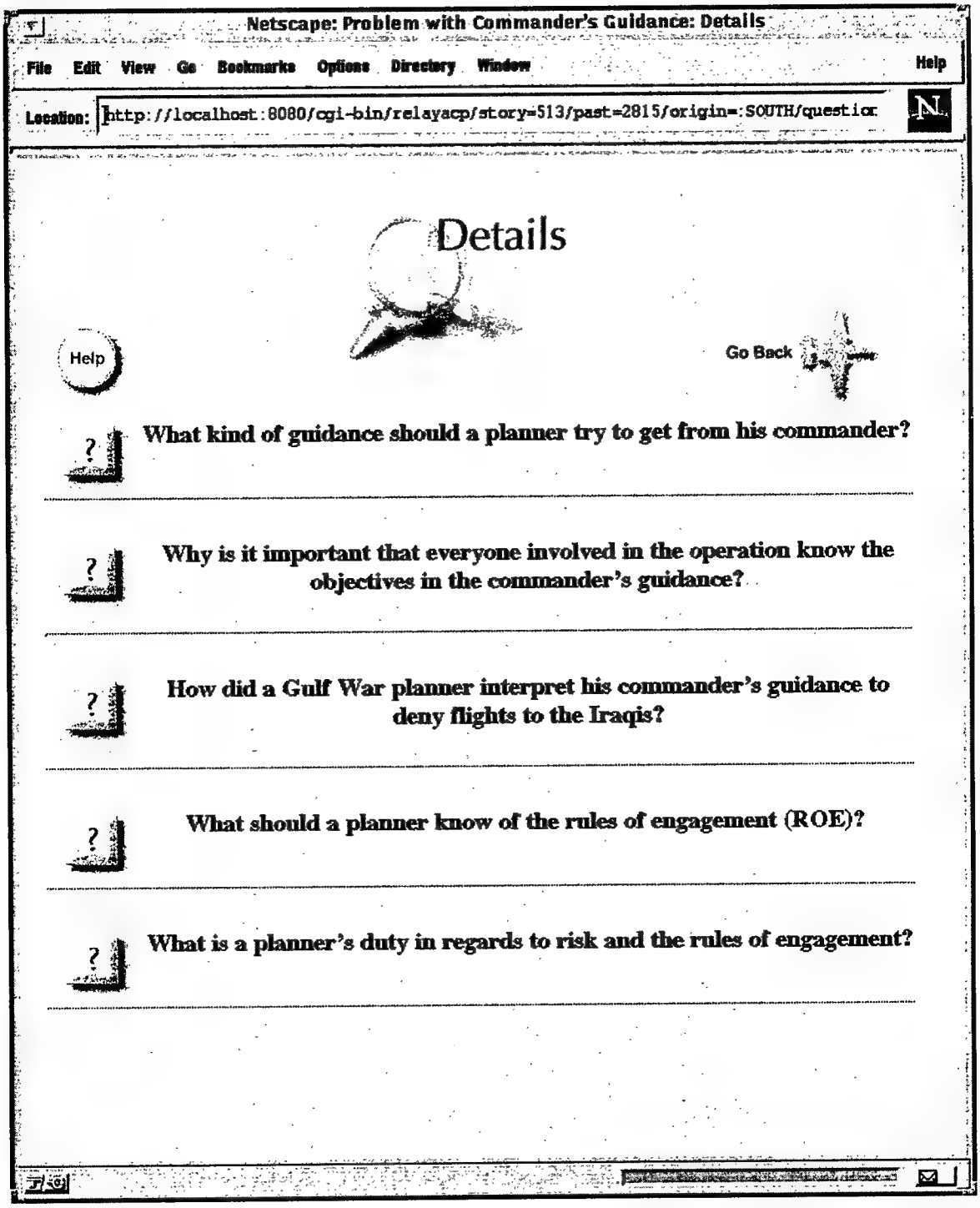

Figure 50: Details Questions

Appendix B: Problems Addressed in Air Campaign Planning Advisor

The ACP task model described in section 24 above provides an entry into the ACPA ASK system. The video content of the ASK system describes, among other things, problems encountered in each of the planning tasks. Although not explicitly represented in the current task model, these problems and methods for dealing with them are an important part of the planning task. This appendix lists the set of problems, associated with each task, for which ACPA provides advice.

1. Obtain Political and Military Objectives

- Commander's guidance is vague.
- Commander's guidance may not contain clear desired end states.
- Commander's guidance expressed in political rather the military end states.
- Commander's guidance may be overly restrictive.
- Commander's guidance may not be provided to planners.
- Planners may not be able to find a military solution to political objectives.
- Military solution may involve unacceptable costs or risks.
- Plan may not meet the CINC's objective.

2. Assess overall combat situation

- Planners may not do situation assessment.
- Planners may not understand importance or utility of situation assessment.
- Planners may face serious time constraints which they feel prevents them from doing situation assessment.
- Planners may not know how best to organize information in a situation assessment.
- Planners may have too much intelligence data to digest in time.
- Planners may not know how to ask intelligence for needed information.
- Planners may not know or have access to the recent history of the enemy forces.

3 Create Prioritized Air Objectives

3.1. Identify COGs

- Planners may skip COG identification stage entirely.
- Planners may not understand what a COG is.
- Planners may not be able to identify critical nodes.
- Planners may confuse COG with vulnerability.
- Planners may not be able to find a COG.
- Planners may identify wrong COGs.
- Planner may fail to use situation assessment to form COGs.
- Planners may not have a procedure for creating COGs.
- Planners may not understand how to derive COGs from situation assessment and political objectives.
- Planners may encounter pressure to service a set of non COG targets.
- Planners may fail to identify Blue and Gray COGs.
- Planners may have made mistake or oversight in situation assessment.

- Planners may not know the enemy's back up capacity.
- Planners may not be able to find a target for a COGs.
- Planners may be prevented by guidance from striking COGs.
- Planners may lack necessary intelligence to identify COGs.

3.2. Project Red and Gray Courses of Action

- Planners may not understand what a COA is.
- Planners may not know how to create a COA.
- Planners may mot understand the importance of a COA analysis.
- Planners may fail to anticipate enemy actions.
- Planners may not understand the perspective or goals of the enemy.
- Planners may not know how to present COA.
- Planners may fail to analyze neutral (Gray) Courses Of Action and their effect on friendly (Blue) and enemy (Red) forces.
- Planners may fail to understand how Red COA effects Blue and Gray COA.
- Planners may place too much importance on Red COA at the expense of creating the campaign plan.
- Planners may be taken by surprise by enemy action.

3.3. Produce candidate air objectives

- Air objectives may not be attached to a CINC objectives or political objectives.
- Air objectives may not be specified to enough detail.
- Air objectives may not contain enough information to be carried out.
- Higher level objectives may not have air objectives that fulfill them.
- Air objectives may be attached to CINC objectives they do not satisfy.
- Planners may not know how to use ACPT to create air objectives.

3.4. Assign measure of merit to each objective

- Air objectives may not have metrics.
- Metrics might be damage rather then effects based.
- Metrics may not be measurable.
- Planners may not know how to attach metric to air objectives in the ACPT.

3.5. Prioritize air objective list

- Planners may not know of understand assets available on any given day, therefore fail to understand the length of a phase.
- Planners may not have any sense of which objectives are more important then other objectives.
- Planners may not know how to determine priorities.
- Planners may not know how to express priorities of objectives.
- Planners may not know use the ACPT to prioritize.

3.6. Brief JFACC

- Briefing may not contain all critical information that needs to be in plan.
- Planners may not know how to present information to JFACC.
- Planners may not understand what information should be and not be in briefing.
- JFACC may make critical changes in plan.
- Planners may not know how to use ACPT briefing tool.

4. Create prioritized task objective

4.1. Produce task objectives for each air objective

- Task objectives might not be attached to air objective.
- Air objectives may exist that have not task objective.
- Task objectives may be attached to air objectives they do not satisfy.
- Planners may not know how to use ACPT to create task objectives.

4.2. Prioritize task objectives

- Task lists may not be prioritized.
- Less important tasks may receive higher priority then more important ones.
- Planners may not know why certain tasks should have higher priorities.
- Planners may not know how to express prioritization.
- Planners may not know how to use the ACPT to prioritize.

4.3. Brief JFACC

- Briefing may not contain all critical information that needs to be in brief.
- Planners may not know how to present information to JFACC.
- Planners may not understand what information should be and not be in briefing.
- JFACC may make critical changes in plan.
- Planners may not know how to use ACPT briefing tool.

4.4. Modify tasks and/or priorities

- Plan may not be in form that can be presented to others.
- Planner may not be able to get feedback on plan.
- Planner may not understand feedback.

4.5. Sequence task objectives

- Planners do not understand how to sequence task.
- Planner may not know what tools are available for sequencing.

5. Create target list

5.1. Assign targets for each task

- Planner may not have enough intelligence to find targets.
- Targets may not support tasks.
- Planners may not understand assigning targets in ACPT.

5.2 Simulate & evaluate (CTEM)

- There may be an data input error into CTEM.
- Planner may doubt accuracy of CTEM output.

7. Assess mission outcome

- No feedback on mission effectiveness provided.
- Intelligence agencies use BDA rather the effects based assessment of mission success.
- Intelligence agencies provide overly conservative estimates of mission effectiveness.
- BDA is conducted with technologies which give an inaccurate picture.

Acknowledgments

The model of ACP described in this paper, and the content of the ACPA system, ire based on interviews with planning experts from Checkmate and the Air Command Staff College, and partly on the strategy-to-tasks model implicit in ISX's Air Campaign Planning Tool (ACPT). Experts we have interviewed include:

- Col. John Warden (retired)
- Col. David Deptula
- Col. Bob Plebanek
- Col. Maris McCrabb
- Maj. Daniel Bakke
- Maj. Randall Peterson
- Maj. Steven Brunhaver
- Maj. Jim Riggins
- Maj. Mark Alred
- Joe Roberts (ISX Corp)

We thank Anna Griffith, Rocky Kendall, Joe Roberts, and their colleagues at ISX Corporation for making it possible for us to interview these experts. Thanks to Marek Lugowski, Rene Rivera, and Eric Johnson for programming, to Alan Libby, Scott Macquarrie and Bob Hooker for content analysis and indexing, to Mark Schaefer and his crew for video production, and to Ruth Schmidt for graphic design. Thanks also to Doug Holmes, Gary Edwards, Alex Kass, Andy Fano, Chris Riesbeck, and Ian Horswill for many helpful discussions on these topics. Finally, we also thank our colleagues in the ARPI Modeling Cluster for sharing their results, and helping us to construct and sharpen our own model. This work was supported in part by the Defense Advanced Research Projects Agency and Rome Laboratory under contract F30602-95-1-0019. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting, part of The Arthur Andersen Worldwide Organization.

Acronyms Used in This Report

ACPA	Air Campaign Planning Advisor
ACPT	Air Campaign Planning Tool
API	Application Program Interface
ATO	Air Tasking Order
BDA	Bomb Damage Assessment
CGI	Common Gateway Interface
CINC	Commander In Chief
COA	Course Of Action
COG	Center Of Gravity
CTEM	Conventional Targeting Effectiveness Model
DOD	Department Of Defense
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JFACC	Joint Forces Air Component Commander
MAAP	Master Air Attack Plan
MOM	Measure Of Merit
URL	Uniform Resource Locator

Bibliography

Agre, P. and Chapman, D. (1987), Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87,* pp. 268-272.

Air Command Staff College (1993), Joint Doctrine Air Campaign Course Air Campaign Planning Handbook.

Bareiss, R. and Osgood, R. (1993), Applying AI Models to the Design of Exploratory Hypermedia Systems. In *Proceedings of Hypertext 93.*

Birnbaum, L., Collins, G., Freed, M., and Krulwich, B. (1990), Model-based diagnosis of planning failures. *Proceedings of AAAI-90,* pp. 318-323.

Brooks, R. (1991), Intelligence without representation. In *Artificial Intelligence,* vol. 47, pp. 139-159.

Ferguson, W., Bareiss, R., Birnbaum, L. and Osgood. R., (1992), ASK Systems: An Approach to the Realization of Story-Based Teachers. In *The Journal of the Learning Sciences,* Vol. 2 No 1, pp 95-134.

Giles, Major P.K., and Galvin, Captain T.P. (1995), Center of Gravity: Determination, Analysis, and Application. Center for Strategic Leadership, U.S. Army War College, Carlisle Barracks, PA 17013

ISX Corporation, (1995), JFACC Planning Tool (JPT) Software User Guide, version 0.1.

ISX Corporation, (1995), JFACC Planning Tool (JPT) Software Reference Manual, version 0.1.

Veshosky, Lt Col G.F., (1995), Combat Air Forces Concept of Operations for the Joint Forces Air Component Commander Planning Tool, ISX Corporation.

Warden, Col J.F. III, (1989), The Air Campaign: Planning for Combat. Pergamon Press, Washington D.C.

United States Department of Defense (1994), Joint Pub 3-56.1 Command and Control for Joint Air Operations.

United States Department of Defense (1995), Joint Doctrine Capstone and Keystone Primer.

United States Department of Defense (1995), Joint Pub 5-0 Doctrine for Planning Joint Operations.

☆U.S. GOVERNMENT PRINTING OFFICE: 1999-610-130-81119

DISTRIBUTION LIST

addresses	number of copies
DR. MICHAEL L. MCHALE AFRL/IFTB 525 BROOKS ROAD ROME, NY 13441-4505	1
NORTHWESTERN UNIVERSITY THE INSTIT FOR LEARNING SCIENCES 1890 MAPLE AVENUE EVANSTON, IL 60208	5
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ATTN: NAN PFRIMMER IIT RESEARCH INSTITUTE 201 MILL ST. ROME, NY 13440	1
AFIT ACADEMIC LIBRARY AFIT/LDR, 2950 P.STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765	1
AFRL/HESC-TDC 2698 G STREET, BLDG 190 WRIGHT-PATTERSON AFB OH 45433-7604	1

ATTN: SMDC IM PL
US ARMY SPACE & MISSILE DEF CMD
P.O. BOX 1500
HUNTSVILLE AL 35807-3801 1

COMMANDER, CODE 4TL000D
TECHNICAL LIBRARY, NAWC-WD
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100 1

CDR, US ARMY AVIATION & MISSILE CMD
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSAM-RD-OB-R, (DOCUMENTS)
REDSTONE ARSENAL AL 35898-5000 2

REPORT LIBRARY
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545 1

ATTN: D'BORAH HART
AVIATION BRANCH SVC 122.10
FOB10A, RM 931
800 INDEPENDENCE AVE, SW
WASHINGTON DC 20591 1

AFIWC/MSY
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016 1

ATTN: KAROLA M. YOURISON
SOFTWARE ENGINEERING INSTITUTE
4500 FIFTH AVENUE
PITTSBURGH PA 15213 1

USAF/AIR FORCE RESEARCH LABORATORY
AFRL/VSOSA(LIBRARY-BLDG 1103)
5 WRIGHT DRIVE
HANSCOM AFB MA 01731-3004 1

ATTN: EILEEN LADUKE/D460
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730 1

OUSD(P)/DTSA/DUTD 1
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

SOFTWARE ENGR'G INST TECH LIBRARY 1
ATTN: MR DENNIS SMITH
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213-3890

USC-ISI 1
ATTN: DR ROBERT M. BALZER
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292-6695

KESTREL INSTITUTE 1
ATTN: DR CORDELL GREEN
1801 PAGE MILL ROAD
PALO ALTO CA 94304

ROCHESTER INSTITUTE OF TECHNOLOGY 1
ATTN: PROF J. A. LASKY
1 LOMB MEMORIAL DRIVE
P.O. BOX 9887
ROCHESTER NY 14613-5700

AFIT/ENG 1
ATTN:TOM HARTRUM
WPAFB OH 45433-6583

THE MITRE CORPORATION 1
ATTN: MR EDWARD H. BENSLEY
BURLINGTON RD/MAIL STOP A350
BEDFORD MA 01730

UNIV OF ILLINOIS, URBANA-CHAMPAIGN 1
ATTN: ANDREW CHIEN
DEPT OF COMPUTER SCIENCES
1304 W. SPRINGFIELD/240 DIGITAL LAB
URBANA IL 61801

HONEYWELL, INC. 1
ATTN: MR BERT HARRIS
FEDERAL SYSTEMS
7900 WESTPARK DRIVE
MCLEAN VA 22102

SOFTWARE ENGINEERING INSTITUTE 1
ATTN: MR WILLIAM E. HEFLEY
CARNEGIE-MELLON UNIVERSITY
SEI 2218
PITTSBURGH PA 15213-38990

UNIVERSITY OF SOUTHERN CALIFORNIA 1
ATTN: DR. YIGAL ARENS
INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY/SUITE 1001
MARINA DEL REY CA 90292-6695

COLUMBIA UNIV/DEPT COMPUTER SCIENCE 1
ATTN: DR GAIL E. KAISER
450 COMPUTER SCIENCE BLDG
500 WEST 120TH STREET
NEW YORK NY 10027

SOFTWARE PRODUCTIVITY CONSORTIUM 1
ATTN: MR ROBERT LAI
2214 ROCK HILL ROAD
HERNDON VA 22070

AFIT/ENG 1
ATTN: DR GARY B. LAMONT
SCHOOL OF ENGINEERING
DEPT ELECTRICAL & COMPUTER ENGRG
WPAFB OH 45433-6583

NSA/OFC OF RESEARCH 1
ATTN: MS MARY ANNE OVERMAN
9800 SAVAGE ROAD
FT GEORGE G. MEADE MD 20755-6000

AT&T BELL LABORATORIES 1
ATTN: MR PETER G. SELFRIDGE
ROOM 3C-441
600 MOUNTAIN AVE
MURRAY HILL NJ 07974

ODYSSEY RESEARCH ASSOCIATES, INC. 1
ATTN: MS MAUREEN STILLMAN
301A HARRIS B. DATES DRIVE
ITHACA NY 14850-1313

TEXAS INSTRUMENTS INCORPORATED 1
ATTN: DR DAVID L. WELLS
P.O. BOX 655474, MS 238
DALLAS TX 75265

KESTREL DEVELOPMENT CORPORATION 1
ATTN: DR RICHARD JULLIG
3260 HILLVIEW AVENUE
PALO ALTO CA 94304

DARPA/ITO 1
ATTN: DR KIRSTIE BELLMAN
3701 N FAIRFAX DRIVE
ARLINGTON VA 22203-1714

NASA/JOHNSON SPACE CENTER 1
ATTN: CHRIS CULBERT
MAIL CODE PT4
HOUSTON TX 77058

SAIC 1
ATTN: LANCE MILLER
144 WESTFIELD
MCLEAN VA 22102

STERLING IMD INC. 1
KSC OPERATIONS
ATTN: MARK MAGINN
BEECHES TECHNICAL CAMPUS/RT 26 N.
ROME NY 13440

HUGHES SPACE & COMMUNICATIONS 1
ATTN: GERRY BARKSDALE
P. O. BOX 92919
BLDG R11 MS M352
LOS ANGELES, CA 90009-2919

SCHLUMBERGER LABORATORY FOR 1
COMPUTER SCIENCE
ATTN: DR. GUILLERMO ARANGO
8311 NORTH FM620
AUSTIN, TX 78720

DECISION SYSTEMS DEPARTMENT 1
ATTN: PROF WALT SCACCHI
SCHOOL OF BUSINESS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CA 90089-1421

SOUTHWEST RESEARCH INSTITUTE 1
ATTN: BRUCE REYNOLDS
6220 CULEBRA ROAD
SAN ANTONIO, TX 78228-0510

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
ATTN: CHRIS DABROWSKI
ROOM A266, BLDG 225
GAITHSBURG MD 20899 — 1

EXPERT SYSTEMS LABORATORY
ATTN: STEVEN H. SCHWARTZ
NYNEX SCIENCE & TECHNOLOGY
500 WESTCHESTER AVENUE
WHITE PLAINS NY 20604 — 1

NAVAL TRAINING SYSTEMS CENTER
ATTN: ROBERT BREAUX/CODE 252
12350 RESEARCH PARKWAY
ORLANDO FL 32826-3224 — 1

DR JOHN SALASIN
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 — 1

DR BARRY BOEHM
DIR, USC CENTER FOR SW ENGINEERING
COMPUTER SCIENCE DEPT
UNIV OF SOUTHERN CALIFORNIA
LOS ANGELES CA 90089-0781 — 1

DR STEVE CROSS
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891 — 1

DR MARK MAYBURY
MITRE CORPORATION
ADVANCED INFO SYS TECH; G041
BURLINTON ROAD, M/S K-329
BEDFORD MA 01730 — 1

ISX
ATTN: MR. SCOTT FOUSE
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE,CA 91361 — 1

MR GARY EDWARDS
ISX
433 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361 — 1

DR ED WALKER 1
BBN SYSTEMS & TECH CORPORATION
10 MOULTON STREET
CAMBRIDGE MA 02238

LEE ERMAN 1
CIMFLEX TEKNOWLEDGE
1810 EMBACADERO ROAD
P.O. BOX 10119
PALO ALTO CA 94303

DR. DAVE GUNNING 1
DARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DAN WELD 1
UNIVERSITY OF WASHINGTON
DEPART OF COMPUTER SCIENCE & ENGIN
BOX 352350
SEATTLE, WA 98195-2350

STEPHEN SODERLAND 1
UNIVERSITY OF WASHINGTON
DEPT OF COMPUTER SCIENCE & ENGIN
BOX 352350
SEATTLE, WA 98195-2350

DR. MICHAEL PITTARELLI 1
COMPUTER SCIENCE DEPART
SUNY INST OF TECH AT UTICA/ROME
P.O. BOX 3050
UTICA, NY 13504-3050

CAPRARO TECHNOLOGIES, INC 1
ATTN: GERARD CAPRARO
311 TURNER ST.
UTICA, NY 13501

USC/ISI 1
ATTN: BOB MCGREGOR
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292

SRI INTERNATIONAL 1
ATTN: ENRIQUE RUSPINI
333 RAVENSWOOD AVE
MENLO PARK, CA 94025

DARTMOUTH COLLEGE
ATTN: DANIELA RUS
DEPT OF COMPUTER SCIENCE
11 ROPE FERRY ROAD
HANOVER, NH 03755-3510 1

UNIVERSITY OF FLORIDA
ATTN: ERIC HANSON
CISE DEPT 456 CSE
GAINESVILLE, FL 32611-6120 1

CARNEGIE MELLON UNIVERSITY
ATTN: TOM MITCHELL
COMPUTER SCIENCE DEPARTMENT
PITTSBURGH, PA 15213-3890 1

CARNEGIE MELLON UNIVERSITY
ATTN: MARK CRAVEN
COMPUTER SCIENCE DEPARTMENT
PITTSBURGH, PA 15213-3890 1

UNIVERSITY OF ROCHESTER
ATTN: JAMES ALLEN
DEPARTMENT OF COMPUTER SCIENCE
ROCHESTER, NY 14627 1

TEXTWISE, LLC
ATTN: LIZ LIDDY
2-121 CENTER FOR SCIENCE & TECH
SYRACUSE, NY 13244 1

WRIGHT STATE UNIVERSITY
ATTN: DR. BRUCE BERRA
DEPART OF COMPUTER SCIENCE & ENGIN
DAYTON, OHIO 45435-0001 1

UNIVERSITY OF FLORIDA
ATTN: SHARMA CHAKRAVARTHY
COMPUTER & INFOR SCIENCE DEPART
GAINESVILLE, FL 32622-6125 1

KESTREL INSTITUTE
ATTN: DAVID ESPINOSA
3260 HILLVIEW AVENUE
PALO ALTO, CA 94304 1

STOLLER-HENKE ASSOCIATES 1
ATTN: T.J. GOAN
2016 BELLE MONTI AVENUE
BELMONT, CA 94002

USC/INFORMATION SCIENCE INSTITUTE 1
ATTN: DR. CARL KESSELMAN
11474 ADMIRALTY WAY, SUITE 1001
MARINA DEL REY, CA 90292

MASSACHUSETTS INSTITUTE OF TECH 1
ATTN: DR. MICHAELE SIEGEL
SLOAN SCHOOL
77 MASSACHUSETTS AVENUE
CAMBRIDGE, MA 02139

USC/INFORMATION SCIENCE INSTITUTE 1
ATTN: DR. WILLIAM SWARTHOUT
11474 ADMIRALTY WAY, SUITE 1001
MARINA DEL REY, CA 90292

STANFORD UNIVERSITY 1
ATTN: DR. GIO WIEDERHOLD
857 SIERRA STREET
STANFORD
SANTA CLARA COUNTY, CA 94305-4125

NCCOSC RDTE DIV D44208 1
ATTN: LEAH WONG
53245 PATTERSON ROAD
SAN DIEGO, CA 92152-7151

SPAWAR SYSTEM CENTER 1
ATTN: LES ANDERSON
271 CATALINA BLVD, CODE 413
SAN DIEGO CA 92151

GEORGE MASON UNIVERSITY 1
ATTN: SUSHIL JAJODIA
ISSE DEPT
FAIRFAX, VA 22030-4444

DIRNSA 1
ATTN: MICHAEL R. WARE
DOD, NSA/CSS (R23)
FT. GEORGE G. MEADE MD 20755-6000

DR. JIM RICHARDSON 1
3660 TECHNOLOGY DRIVE
MINNEAPOLIS, MN 55418

LOUISIANA STATE UNIVERSITY 1
COMPUTER SCIENCE DEPT
ATTN: DR. PETER CHEN
257 COATES HALL
BATON ROUGE, LA 70803

INSTITUTE OF TECH DEPT OF COMP SCI 1
ATTN: DR. JAIDEEP SRIVASTAVA
4-192 EE/CS
200 UNION ST SE
MINNEAPOLIS, MN 55455

GTE/BBN 1
ATTN: MAURICE M. MCNEIL
9655 GRANITE RIDGE DRIVE
SUITE 245
SAN DIEGO, CA 92123

UNIVERSITY OF FLORIDA 1
ATTN: DR. SHARMA CHAKRAVARTHY
E470 CSE BUILDING
GAINESVILLE, FL 32611-6125

AFRL/IFT 1
525 BROOKS ROAD
ROME, NY 13441-4505

AFRL/IFTM 1
525 BROOKS ROAD
ROME, NY 13441-4505

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.